ENCICLOPEDIA PRACTICA DE LA

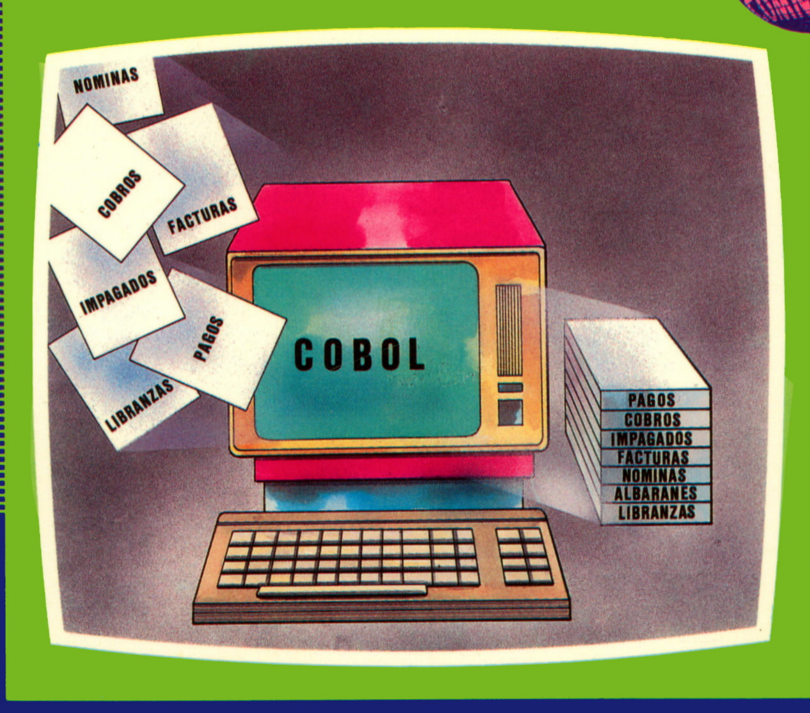
INFORMATICA APLICADA

33

Cobol

AIA

SINCLAIR AMSTRAD COMMODORE IBM



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

33

Cobol

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación
y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo XXXIII. **Cobol**
AULA DE INFORMATICA APLICADA
ANA PASTOR, Licenciada en Informática
ELOY PEREZ, Licenciado en Informática

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta (Ed. Iberia Mart I). Teléf. 810 52 13. 28020 Madrid

Publicidad:
Gofar Publicidad, S.A. San Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-128-2
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 16.872-1987
Printed in Spain - Impreso en España.

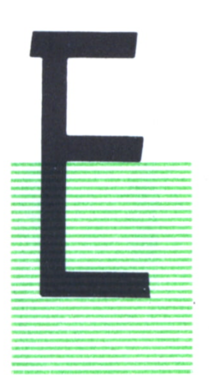
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta (Ed. Iberia Mart I). Teléf. 810 52 13. 28020 Madrid
Junio, 1987
P.V.P. Canarias: 365,-

INDICE

COBOL 1



INTRODUCCION



EL lenguaje de programación más utilizado en el desarrollo de aplicaciones de gestión es el COBOL.

COBOL son las siglas de Common Business Oriented Language (lenguaje común orientado a los negocios). Este lenguaje fue producto de una reunión celebrada en Estados Unidos en 1959, entre fabricantes de ordenadores y usuarios de los mismos.

Al año siguiente de la reunión reseñada anteriormente, surgió la primera versión del lenguaje COBOL, denominándose COBOL-60. Este primer producto fue depurándose, dando lugar a nuevas versiones, como COBOL-61, COBOL-62, etc.

En 1968 se iniciaron los pasos para estandarizar dicho lenguaje. Se ocupaba de ello el American National Standard Institute. De esta forma surgió el ANS COBOL.

Una propiedad importante del COBOL es su legibilidad. Su sintaxis está muy próxima a la gramática inglesa, lo que permite leer un programa escrito en COBOL como si se tratase de un texto escrito en inglés.

Es un lenguaje orientado, como ya se ha dicho antes, a las aplicaciones de gestión por su estructura y diseño; en cambio, no es adecuado para realizar programas que requieran cálculos de expresiones matemáticas complejas.

ESTRUCTURA DE UN PROGRAMA COBOL 2



N programa COBOL se compone de cuatro "DIVISIONES" que siempre deben aparecer en el mismo.

Estas son:

— IDENTIFICATION DIVISION.
— ENVIRONMENT DIVISION.
— DATA DIVISION.
— PROCEDURE DIVISION.

En la primera se refleja el nombre del programa, del autor, la fecha de escritura, etc. Es decir, aporta información adicional al programa.

La **ENVIRONMENT DIVISION** informará al compilador del número y tipo de los ficheros que se van a utilizar.

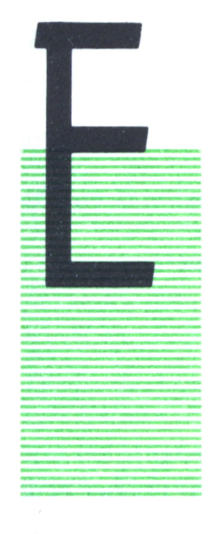
La descripción de los registros de los ficheros y todas las variables que necesita el programa se realiza en la **DATA DIVISION**.

Por último, en la **PROCEDURE DIVISION** es donde se encuentran las instrucciones ejecutables.

Las tres primeras divisiones son de información y definición, siendo la última donde se hallan las instrucciones concretas que solventan el problema que quiere ser abordado.

El nombre de las cuatro divisiones debe aparecer en todo programa COBOL, aunque alguna de ellas (ENVIRONMENT y/o DATA) pueda estar vacía.

IDENTIFICATION DIVISION 3



ESTA primera división de un programa COBOL tiene, como función principal, la de asignar un nombre al programa, además de aportar otra serie de información adicional.

Su estructura es la siguiente:

```
{ IDENTIFICATION DIVISION. }
{ ID DIVISION.             }
    PROGRAM-ID. Nombre del programa.
[DATE-WRITTEN. Fecha de escritura.]
[INSTALLATION. Nombre de la instalación.]
[DATE-WRITTEN. Fecha de escritura.]
[DATE-COMPILED. Fecha de la última compilación.]
[SECURITY. Notas de seguridad.]
[REMARKS. Comentarios.]
```

En la definición de la estructura se utilizan determinados símbolos con significación propia, que se repetirán a lo largo de todo el libro. Estos se explican a continuación.

Las llaves indican que dentro de las opciones que encierran, hay que elegir obligatoriamente una. En este caso, es necesario seleccionar una de las dos formas:

— IDENTIFICATION DIVISION, o
— ID DIVISION.

La cláusula encerrada entre corchetes es opcional, no es obligatorio utilizarla. Refiriéndonos de nuevo al ejemplo, no es obligatorio poner AUTHOR, INSTALLATION, DATE-WRITTEN, etc.

Por último, las palabras subrayadas indican que en caso de selección de esa cláusula, es obligatorio que se encuentren en la misma.

Por la estructura de la IDENTIFICATION se deduce que sólo es obli-

gatorio que aparezcan las cláusulas: IDENTIFICATION DIVISION o ID DIVISION y PROGRAM-ID, siendo el resto opcional.

El nombre del programa debe aparecer después de PROGRAM-ID. Este debe ser, como máximo, de ocho caracteres. Si se utilizan más de ocho caracteres, éstos serán ignorados por el compilador.

El párrafo AUTHOR se utiliza para introducir el nombre del programador.

En INSTALLATION se refleja en qué instalación ha sido desarrollado el programa.

La fecha de redacción del programa aparecerá en DATE-WRITTEN.

Detrás de DATE-COMPILED mostrará la última fecha de compilación del programa. Esta es escrita por el propio compilador, sustituyendo la que hubiese antes.

Los mensajes de seguridad sobre uso o manipulación del programa se encuentran en el párrafo encabezado por SECURITY.

Si se desea introducir un grupo de comentarios se realizará en REMARKS.

Un ejemplo de IDENTIFICATION DIVISION es el siguiente:

```
IDENTIFICATION DIVISION
PROGRAM-ID. PEDIDOS.
AUTHOR. E.P.H.
INSTALLATION. CENTRO DE CALCULO DE AIA.
DATE-WRITTEN. MARZO 1987.
DATE-COMPILED.
SECURITY. ESTE PROGRAMA SOLO PUEDE SER UTILIZADO POR
          PERSONAL AUTORIZADO POR AIA.
REMARKS.  PROGRAMA PARA LA CONFECCION DE PEDIDOS DE
          MATERIAL.
```

PRIMER PROGRAMA COBOL 



EN este apartado se van a estudiar los elementos básicos para poder realizar un programa COBOL muy sencillo que sirva para fijar ideas, y como base para seguir avanzando en el estudio de este lenguaje.

El lector ya conoce la estructura de la IDENTIFICATION DIVISION por el apartado anterior.

De momento, no se van a tratar la ENVIRONMENT DIVISION ni la DATA DIVISION. Se verán más adelante.

A continuación se explican dos instrucciones ejecutables del lenguaje COBOL, que nos permitirán escribir nuestro primer programa. El formato de la primera instrucción es:

$$\underline{\text{DISPLAY}} \quad \left[\begin{Bmatrix} \text{literal} \\ \text{campo} \end{Bmatrix} \right]$$

De la estructura del anterior formato se deduce que puede aparecer un DISPLAY solo, o un DISPLAY seguido de uno o varios literales, y uno o varios campos; pudiendo estar entremezclados campos y literales.

Esta estructura COBOL visualiza en la pantalla el mensaje y/o los campos que acompañan al DISPLAY.

Un literal en COBOL es una ristra de caracteres de todos los tipos, de longitud no mayor de 120, todos ellos encerrados entre apóstrofos.

Todo programa debe tener un final. En COBOL la instrucción que permite parar la ejecución es:



Por último, vamos a definir lo que es un nombre de párrafo.

Se llama nombre de párrafo a un nombre creado por el programador que agrupa a un conjunto de instrucciones COBOL.

Es importante que ese nombre de párrafo denote lo que realizan las instrucciones que agrupa.

La longitud máxima de un nombre de párrafo es de 30 caracteres.

Ya se está en disposición de escribir un programa COBOL. Un ejemplo se muestra a continuación.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. INICIAL.
 AUTHOR. EPH.
 DATE-WRITTEN. ABRIL-87.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

INICIO-PROGRAMA.
        DISPLAY 'ESTE ES MI PRIMER PROGRAMA COBOL'.
        DISPLAY 'NO ME HA RESULTADO DIFICIL ESCRIBIRLO'.

FIN-PROGRAMA.
        STOP RUN.
```

Se puede ver en el ejemplo cómo están reflejadas las cuatro divisiones, aunque dos de ellas estén vacías porque para este programa no son necesarias.

La IDENTIFICATION DIVISION tiene alguno de los párrafos opcionales. En los programas que se hagan a continuación sólo pondremos la cláusula obligatoria PROGRAM-ID, no escribiendo el resto, que son optativas; pero se debe recalcar que es conveniente escribir, en todos los programas que se hagan, el autor y la fecha en que se han escrito. En la PROCEDURE DIVISION se distinguen dos nombres de párrafo. El primero de ellos agrupa a las dos instrucciones DISPLAY.

Muchos compiladores obligan a que aparezca siempre un nombre de párrafo después de la cláusula PROCEDURE DIVISION.

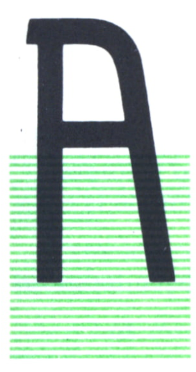
El segundo nombre de párrafo contiene la instrucción de final de ejecución del programa.

El ejemplo contiene dos instrucciones DISPLAY con un literal cada una de ellas.

La ejecución de este programa mostrará por pantalla los dos literales en sendas líneas, una por cada DISPLAY.

Como se puede comprobar, no es complicado escribir un programa en COBOL.

MARGENES Y PUNTUACION 5



A diferencia de otros lenguajes, la sintaxis en la escritura de un programa COBOL está sometida a unas normas rígidas.

Echando una rápida mirada al ejemplo anterior, se observa que hay sentencias que están más a la derecha que otras; esto no es por capricho, sino porque cada tipo de sentencia debe comenzar en su correspondiente "MARGEN".

Imagínese que está sentado delante de la pantalla de su ordenador, dispuesto a introducir el ejemplo anterior. El ancho de la línea de pantalla es de 80 caracteres. Vamos a ver a continuación cómo distribuye el COBOL esos 80 caracteres.

Los 6 primeros se pueden utilizar para numerar las líneas. Esto era muy frecuente realizarlo cuando se trabajaba con fichas perforadas. Era conveniente numerarlas por si ocurría la "desgracia" de que el taco de tarjetas se cayera y se descolocase por completo. Si las fichas estaban numeradas, la labor de "reconstrucción" del programa era menos ardua.

Pero con la implantación de terminales, a través de los cuales se introducen y modifican programas, la numeración de sentencias ha caído en desuso, dejándose esas seis columnas en blanco.

La columna 7 tiene un significado especial, que veremos a continuación.

Las columnas de la 8 a la 11 componen el "MARGEN A".

En el margen A debe aparecer al menos el primer carácter del nombre de todas las divisiones, todas las cláusulas que forman la IDENTIFICATION DIVISION y todos los nombres de párrafos.

Las columnas desde la 12 a la 72 son las que forman el "MARGEN B". En éste se encuentran todas las instrucciones que aparecen en la PROCEDURE DIVISION.

A medida que vayan apareciendo nuevas sentencias se irán enmarcando en el margen adecuado.

Cuando se dice que una sentencia debe comenzar en el margen A, puede utilizarse para ello cualquiera de las 4 columnas que lo forman. Algunos programadores acostumbran a poner el nombre de las divisiones en la columna 8 y las cláusulas que la componen, que deben encuadrarse también en el margen A, en las columnas siguientes. Esto da un efecto de jerarquía que puede facilitar la labor de lectura del programa.

Esto también es válido para el margen B.

Las columnas que van desde la 73 a la 80 son ignoradas por el compilador, es decir, en ellas puede colocarse cualquier cosa. Por tanto, toda instrucción **COBOL** puede ocupar como máximo hasta la columna 72.

Por último, queda por explicar el significado de la columna 7.

En ésta pueden aparecer tres caracteres:

— Un blanco que denota que es una línea normal.

— "*", el asterisco significa que la línea es de comentario. El programador podrá insertar en ella lo que quiera, el compilador lo ignorará.

— "-", el guión indica que la línea actual es continuación de la anterior.

A continuación se presenta un ejemplo en el que se reflejan todos los aspectos explicados en este apartado.



Se puede ver en este ejemplo una instrucción que es un comentario (en la columna 7 un asterisco).

También existe una instrucción DISPLAY que ocupa dos líneas.

El proceso consiste en finalizar de poner el contenido del literal cuando se llegue a la posición 72, sin cerrar el apóstrofo. En la línea siguiente poner en la columna 7 un guión, colocar un apóstrofo en cualquier columna del margen B, continuar el literal y cuando acabe poner otro apóstrofo.

Si necesitase más líneas de continuación se repetiría el proceso tantas veces como fuese necesario. El número máximo de líneas de continuación está en función del compilador utilizado; suele ser de 7 líneas.

Como puede observarse en los dos ejemplos presentados, todas las instrucciones acaban en punto. Esto es otra norma que ha de cumplirse. Sólo es opcional en las instrucciones dentro de la PROCEDURE DIVISION, en las que puede no ponerse, excepto en los nombres de párrafo y en las instrucciones inmediatamente superiores que preceden a estos nombres de párrafo.

También existen otras normas de puntuación que pueden resumirse en las siguientes reglas:

— Los caracteres coma, punto y coma y punto, no pueden ir precedidos de un espacio en blanco, pero deben ir seguidos al menos por uno.

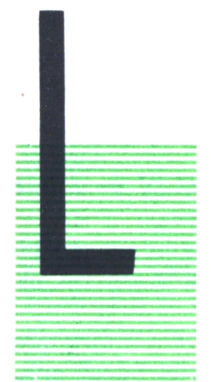
— El paréntesis izquierdo no puede ir seguido de un espacio.

— El paréntesis derecho no debe ir precedido de un espacio.

— El signo igual y los operadores aritméticos deben estar precedidos y seguidos por un blanco.

Estas reglas siempre deben tenerse en cuenta durante la confección de un programa COBOL, porque si no se hace así, los errores de compilación serán abundantes.

CAMPOS VARIABLES 6



O habitual en cualquier programa es utilizar campos variables, cuyo valor se vaya modificando a lo largo de la ejecución del mismo.

Estos deben definirse dentro de una sección de la DATA DIVISION. Esta tiene dos secciones:

— FILE SECTION.
— WORKING-STORAGE SECTION.

Ambas se encuadran en el margen A.

La FILE SECTION se estudiará junto a la ENVIRONMENT DIVISION porque están íntimamente relacionadas.

La definición de todos los campos de trabajo se realiza en la WORKING-STORAGE SECTION.

Para describir un campo es necesario conocer el concepto de número de nivel y formato del campo.

Se van a explicar estos conceptos sobre el siguiente ejemplo, que muestra la parte correspondiente a la DATA DIVISION de un programa.

```
DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 NOMBRE                PIC X(15).

01 FECHA.
   05 DIA                PIC 99.
   05 MES                PIC 99.
   05 AGNO               PIC 99.

01 IDENTIFICACION.
```

```
05 DNI                    PIC 9(8).
05 NOMINAL.
   10 NOMBRE              PIC X(15).
   10 APELLIDOS           PIC X(30).
05 DIRECCION.
   10 CALLE     PIC X(23).
   10 NUMERO              PIC X(3).
   10 COD-POST            PIC X(5).
```

A la izquierda del nombre de los campos se observa que aparecen unos números (01, 05 y 10). Estos son los números de nivel. Pueden tomar valores entre 01 y 49. Con los números de nivel se establece una jerarquía dentro de la estructura de cada dato.

El nombre de campo calificado con un número de nivel 01 es el que globaliza todos los subcampos, en el caso de que los hubiera.

En la primera línea de la WORKING del ejemplo se describe un campo individual (que no está subdividido) y, por tanto, está calificado con un número de nivel 01.

El número de nivel 01 debe estar en el margen A, pero el nombre que le sigue debe comenzar en el margen B. Para el resto de niveles puede aplicarse la regla anterior o, lo que es más habitual, comenzar en el margen B.

En la descripción del segundo campo vemos que éste es un campo subdividido. Cualquier fecha puede subdividirse en día, mes y año. En el ejemplo que se presenta esta división se ha producido.

Hay un campo global que es FECHA con nivel 01, y tres subcampos que componen el anterior: DIA, MES y AGNO con nivel 05.

Al ver un número de nivel mayor que 01, significa que éste forma parte del número de nivel inferior inmediatamente anterior. En este caso los tres subcampos dependen directamente de FECHA, porque es el único número de nivel inferior a 05 inmediatamente anterior.

Si en el programa se desea acceder sólo al mes, haremos referencia al subcampo MES. Pero si se desea acceder a la fecha en conjunto, se utilizará el campo global FECHA, que contiene la fecha al completo. El siguiente campo que aparece en la WORKING es IDENTIFICACION. Como puede verse, éste tiene tres niveles de división. NOMBRE y APELLIDOS forman el subcampo NOMINAL. DIRECCION está compuesto por CALLE, NUMERO y COD-POST.

Esta jerarquía de los campos viene reflejada por los números de nivel. En el ejemplo, los de nivel 10 dependen del nivel 05, y los tres de nivel 05 (DNI, NOMINAL y DIRECCION) dependen del nivel 01 IDENTIFICACION.

De esta forma se puede ir subdividiendo la información todo lo necesario, con la utilización de los números de nivel.

Detrás de estos niveles aparece el nombre del campo. Estos pueden tener una longitud máxima de 30 caracteres y deben ser letras, números o guiones; siempre que un campo no empiece o acabe con guión, ni tenga como primer caracter un dígito.

El nombre de un campo nunca debe coincidir con una "palabra reservada". Palabras reservadas son todas aquellas que tienen un significado determinado para el compilador. Es decir, un nombre de campo no podría ser PROCEDURE, porque se confundiría con la PROCEDURE DIVISION. Tampoco podría utilizarse SECTION, FILE, etc. A continuación del nombre del campo viene la palabra reservada PIC o PICTURE. Esta denota que los caracteres que vengan detrás definen el tipo de datos que va a contener la variable y su longitud. Ahora se estudiarán tres caracteres que definen otros tantos tipos de variables.

— A. La variable es alfabética. Podrá contener letras y el espacio en blanco.

— X. La variable es alfanumérica. Contendrá cualquier carácter.

— 9. La variable es numérica, su contenido serán dígitos del 0 al 9.

En la WORKING mostrada en la figura 3 se observa que la variable NOMBRE es alfanumérica y su longitud es de 15 caracteres, como indica el número encerrado entre paréntesis que sigue a "X".

Es equivalente poner X(15) o 15 X's seguidas. Siempre se adopta la primera forma, porque es más segura y cómoda.

El subcampo MES es numérico y de longitud 2. El campo FECHA tiene longitud 6, recuerde que agrupa a DIA, MES y AGNO.

APELLIDOS es alfanumérico de longitud 30. El subcampo NOMINAL es alfanumérico y consta de 45 caracteres, mientras que IDENTIFICACION tendrá 84 caracteres de longitud.

INSTRUCCION DE MOVIMIENTO 7



A se sabe cómo definir campos variables en la WORKING-STORAGE SECTION. A continuación se explicarán algunas de las herramientas para utilizar esas variables en la PROCEDURE DIVISION.

El verbo de movimiento es el siguiente:

$$\underline{\text{MOVE}} \quad \left\{ \begin{array}{l} \text{literal} \\ \\ \text{campo-1} \end{array} \right\} \quad \underline{\text{TO}} \text{ campo-2...}$$

El significado de esta instrucción es el siguiente: el *literal* o contenido del *campo-1* pasa al *campo-2*, perdiendo éste el valor que tuviera antes. El valor de *campo-1* no se modifica.

El siguiente programa es una aplicación de la sentencia que se acaba de estudiar.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-MOVES.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 NOMBRE-1                 PIC X(10).
01 NOMBRE-2                 PIC X(4).
01 INICIAL                  PIC X.


PROCEDURE DIVISION.

INICIO.
```

```
           MOVE 'ANTONIO' TO NOMBRE-1.
           MOVE 'JUAN'    TO NOMBRE-2.
           DISPLAY NOMBRE-1.
           MOVE NOMBRE-2  TO NOMBRE-1
                             INICIAL.
           DISPLAY 'LA INICIAL DE ' NOMBRE-1 ' ES ' INICIAL.

       FIN-PROGRAMA.
           STOP RUN.
```

En este ejemplo se han definido tres campos alfanuméricos (NOMBRE-1, NOMBRE-2, INICIAL) de longitudes 10, 4 y 1, respectivamente.

Las dos primeras instrucciones son dos MOVE's que introducen el contenido de dos literales en los campos NOMBRE-1 y NOMBRE-2. El contenido de estos campos después de ejecutarse esas instrucciones será el siguiente:

NOMBRE-1	A	N	T	O	N	I	O
NOMBRE-2	J	U	A	N			

El primer movimiento es de un literal de longitud 7 a un campo de longitud 10. Al pasar al campo, el literal se ajusta a la izquierda y rellena con blancos por la derecha hasta completar la longitud del campo.

En el segundo movimiento el número de caracteres del literal coincide con los que puede contener el campo.

Después se muestra por pantalla el contenido del campo NOMBRE-1 mediante la instrucción DISPLAY. El resultado de ejecutarla será:

ANTONIO

La siguiente instrucción consiste en llevar el contenido del campo NOMBRE-2 (que es "JUAN") a los campos NOMBRE-1 e INICIAL.
El resultado será:

NOMBRE-1	J	U	A	N						
INICIAL	J									

El contenido del NOMBRE-1 antes de ejecutarse la sentencia era "ANTONIO", después del MOVE este valor desaparece y es sustituido por "JUAN", más los blancos de relleno por la derecha. Es decir, después de un MOVE el campo receptor siempre pierde el contenido que tuviese.

En la misma instrucción se hace también un movimiento al campo INICIAL. La longitud de éste es de 1; por tanto, no podrá almacenar todo el contenido de la variable NOMBRE-2. El proceso que se produce es el siguiente: se empieza a ajustar los caracteres por la izquierda hasta completar la longitud del campo receptor, perdiéndose los caracteres que no quepan. Se produce un truncamiento.

El contenido de INICIAL será "J", perdiéndose "UAN".

A diferencia del campo receptor, el emisor, después de ejecutarse una instrucción MOVE, no pierde su valor.

En el ejemplo, después de realizarse la última instrucción de movimiento que se ha tratado, el contenido de NOMBRE-2 seguirá siendo "JUAN".

En el programa, la sentencia siguiente es un DISPLAY compuesto por literales y variables, cuyo resultado será:

LA INICIAL DE JUAN   ES J

Las instrucciones de movimiento no son sólo aplicables a campos alfanuméricos, sino también a numéricos, como muestra el siguiente ejemplo.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-MOVES-2.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 CIFRA-1                   PIC 9(5).
01 CIFRA-2                   PIC 9(2).
01 CIFRA-3                   PIC 9.


PROCEDURE DIVISION.

INICIO.
        MOVE 2834      TO CIFRA-1.
        MOVE 32        TO CIFRA-2.
        DISPLAY CIFRA-1 ' ' CIFRA-2.
        MOVE CIFRA-2 TO CIFRA-1
                        CIFRA-3.
        DISPLAY CIFRA-1 ' ' CIFRA-2 ' ' CIFRA-3.

FIN-PROGRAMA.
        STOP RUN.
```

Este programa es análogo al anterior, diferenciándose sólo en que éste utiliza campos numéricos en lugar de alfanuméricos.

Las dos primeras instrucciones consisten en mover el contenido de dos literales numéricos (no están encerrados entre apóstrofos) a otros tantos campos numéricos.

El resultado de estos movimientos es el siguiente:

CIFRA-1	0 2 8 3 4
CIFRA-2	0 3 2

En el movimiento numérico, al contrario que en los alfanuméricos, se ajustan los campos por la derecha, rellenando con ceros no significativos por la izquierda.

El resultado después de ejecutarse el primer DISPLAY será:

02834 032

El contenido de los campos CIFRA-2 y CIFRA-3 después de ejecutarse la siguiente instrucción de movimiento será:

CIFRA-1	0 0 0 3 2
CIFRA-3	2

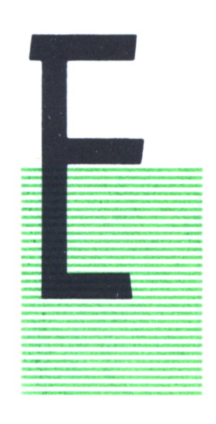
Al igual que antes, el contenido anterior del campo receptor desaparece y el del emisor permanece. En CIFRA-1 se ha introducido el contenido de CIFRA-2, ajustando por la derecha y rellenando a ceros por la izquierda.

La longitud de CIFRA-3 es menor que la del campo emisor; por tanto, se producirá truncamiento. Comienzan a colocarse las cifras por la derecha, perdiéndose las que no quepan, en este caso el 3.

Por último, hay que reseñar que un campo numérico puede ser movido a uno alfanumérico, pero no al revés, porque se producirá un error de compilación.

Hay que tener cuidado con el tipo del campo emisor y receptor para que no surjan problemas como éste.

CONSTANTES FIGURATIVAS 8



XISTE un grupo de constantes que tienen un valor predefinido por el COBOL. Se conocen como constantes figurativas y son las siguientes:

— ZERO: representa el valor cero o ceros. Puede asignarse a campos alfanuméricos y numéricos.

— ZEROS o ZEROES: es la forma plural de ZERO y equivalente a ella.

— SPACE o SPACES: representan uno o varios espacios en blanco.

— QUOTE o QUOTES: denotan uno o más apóstrofos.

— ALL “carácter”: representa cualquier aparición del juego de caracteres que maneje el ordenador, excepto el apóstrofo. El carácter que se repite es el que está encerrado entre apóstrofos siguiendo a la palabra reservada ALL.

— HIGH-VALUE o HIGH-VALUES: representa uno o más sucesos del valor más elevado del código de caracteres usado.

— LOW-VALUE o LOW-VALUES: denota uno o más sucesos del valor más bajo del código de caracteres utilizado.

La utilización de alguna de estas constantes, así como un método para inicializar variables en la WORKING-STORAGE, se presentan en el siguiente programa:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EJ-CONSTANTES.

      *
      * Este programa escribe un nombre y apellidos encerrados
      * entre apóstrofos, subrayando los apellidos.
      *
```

```
ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 IDENTIFICACION.
   05 FILLER              PIC X          VALUE QUOTE.
   05 APELLIDO-1          PIC X(13).
   05 FILLER              PIC X          VALUE SPACES.
   05 APELLIDO-2          PIC X(13).
   05 FILLER              PIC XX         VALUE ', '.
   05 NOMBRE              PIC X(10).
   05 FILLER              PIC X          VALUE QUOTE.

01 SUBRAYADO.
   05 FILLER              PIC X          VALUE SPACES.
   05 FILLER              PIC X(13)      VALUE ALL '-'.
   05 FILLER              PIC X(13)      VALUE ALL '-'.


PROCEDURE DIVISION.

INICIO.
       MOVE 'LAURA'    TO NOMBRE.
       MOVE 'PASTOR'   TO APELLIDO-1.
       MOVE 'URTIAGA'  TO APELLIDO-2.
       DISPLAY IDENTIFICACION.
       DISPLAY SUBRAYADO.

FIN-PROGRAMA.
       STOP RUN.
```

En la **WORKING-STORAGE** encontramos dos grupos, uno conteniendo el nombre y apellidos, y el otro el subrayado de los apellidos.

En la descripción de estos dos grupos se observa que el nombre de campo **FILLER** se repite varias veces. A todos los campos que no van a ser referenciados en la **PROCEDURE** se les llama de esta forma, ahorrándose la creación de un nombre de datos nuevo. Estos campos, la gran mayoría de las veces, se utilizan para separar campos que van a ser escritos. En el ejemplo se utilizan para separar el nombre, primer apellido y segundo apellido.

También aparece la cláusula **VALUE**. Esta se utiliza para dar valor al campo que califica. El valor es la constante que le sigue.

El primer subcampo de **IDENTIFICACION** es un **FILLER**, que tiene un valor inicial de **QUOTE**, es decir, un apóstrofo.

Si no existiese esta constante figurativa no podrían reflejarse apóstrofos, ya que en un programa, si aparece la cláusula:

05 FILLER PIC X VALUE '''.

el compilador mostrará un error en esa línea, ya que el segundo apóstrofo lo tomará como final de literal, no como contenido del mismo y al detectar el tercero dará un error. Este problema se soluciona utilizando la constante QUOTE.

El siguiente campo es APELLIDO-1, al que no se le da ningún valor inicial, porque lo tomará en la PROCEDURE. Lo mismo ocurre con APELLIDO-2 y NOMBRE.

Para separar APELLIDO-1 de APELLIDO-2 se utiliza un FILLER inicializado con un blanco, utilizando la constante SPACE. Esta constante se puede sustituir por una cadena de espacios en blanco acotados por apóstrofos, pero es más cómodo utilizar la forma SPACE porque no es necesario contar cuántos blancos se ponen.

El siguiente FILLER es de longitud 2 y está inicializado por un literal no numérico formado por una coma y un espacio en blanco. Hay que tener cuidado de no poner mayor longitud en el literal que sigue a VALUE que en el campo que inicializa.

El último FILLER es para poner el apóstrofo final que delimita el nombre y apellidos.

El siguiente campo fija el subrayado de los apellidos. Como no contiene ningún campo que aparezca en la PROCEDURE, está compuesto por FILLER's.

Dos de estos están inicializados con espacios para ajustar el subrayado a la posición de los apellidos en el grupo anterior.

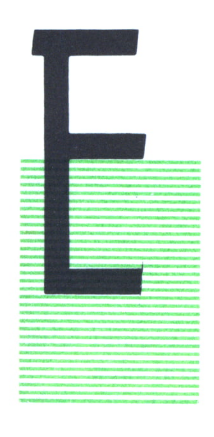
Los dos campos de subrayado, cuya longitud coincide lógicamente con la de los apellidos. Para inicializarlo se ha utilizado la constante ALL, seguida del carácter "-". El efecto de éste es llenar los 13 caracteres del FILLER con el guión, es decir, rellena la totalidad del campo con el carácter encerrado entre apóstrofos.

El resultado de ejecutar este programa será:

'PASTOR URTIAGA , LAURA '

La cláusula VALUE puede ser sustituida por MOVE's en la PROCEDURE, pero es mucho más cómodo hacerlo en la WORKING. Además, si se hace con MOVE's, cada subcampo de datos deberá tener un nombre diferente, no pudiendo utilizar la técnica de los FILLER's.

LECTURA DINAMICA 9



EN todos los programas que se han presentado hasta ahora los datos se han introducido con **MOVE's**. Esto conlleva que si se quiere introducir datos diferentes debe modificarse el programa, siendo esta solución absurda e inviable.

La solución es que el programa tenga la capacidad de pedir los datos al usuario, creándose de esta forma una independencia entre los datos y el programa.

La instrucción para leer datos de pantalla es:

<u>ACCEPT</u> nombre-campo.

El programa que se presenta a continuación es análogo al anterior, únicamente se incluye la lectura dinámica de datos.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-ACCEPT.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 IDENTIFICACION.
   05 FILLER              PIC X          VALUE QUOTE.
   05 NOMBRE              PIC X(10).
   05 FILLER              PIC XX         VALUE ', '.
   05 APELLIDO-1          PIC X(13).
   05 FILLER              PIC X          VALUE SPACES.
   05 APELLIDO-2          PIC X(13).
   05 FILLER              PIC X          VALUE QUOTE.
```

```
01 SUBRAYADO.
   05 FILLER              PIC X(13)       VALUE SPACES.
   05 FILLER              PIC X(13)       VALUE ALL '-'.
   05 FILLER              PIC X(13)       VALUE ALL '-'.

PROCEDURE DIVISION.

INICIO.
       DISPLAY 'PRIMER APELLIDO: '.
       ACCEPT APELLIDO-1.
       DISPLAY 'SEGUNDO APELLIDO: '.
       ACCEPT APELLIDO-2.
       DISPLAY 'NOMBRE: '.
       ACCEPT NOMBRE.
       DISPLAY IDENTIFICACION.
       DISPLAY SUBRAYADO.

FIN-PROGRAMA.
       STOP RUN.
```

Este programa ya no trata datos fijos, sirve para captar cualquier nombre y apellidos, escribirlos en pantalla con los apellidos subrayados.

La primera instrucción es un DISPLAY para indicar al usuario que debe introducir el primer apellido.

Después el ordenador se queda parado a la espera de que se teclee el primer apellido y pulse la tecla [↵] para que el ordenador tome los datos introducidos.

Esta secuencia podría ser:

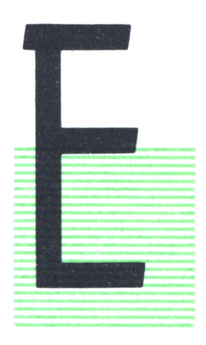
PRIMER APELLIDO
PASTOR [↵]

En ese momento, el contenido de APELLIDO-1, que con el primer ACCEPT será "PASTOR", toma el valor que haya sido tecleado.

El mismo proceso se repite con el resto de DISPLAY's y ACCEPT's.

INSTRUCCIONES ARITMETICAS





L lenguaje COBOL, al ser un lenguaje destinado al mundo de la gestión, no incluye entre sus instrucciones la posibilidad de evaluar operaciones aritméticas complejas.

Es entonces el momento de recordar a aquellos lectores que piensen en complicados cálculos trigonométricos e integrales que el COBOL no es su lenguaje.

Por el contrario, si lo que se necesita es un diseño de salida de fácil lectura y agradable a la vista, cuya obtención no requiera cálculos excesivamente difíciles, el COBOL dispone de todas las herramientas precisas; entre ellas se encuentran instrucciones como:

— ADD.
— SUBTRACT.
— MULTIPLY.
— DIVIDE.



SUMAR

El formato de la instrucción sumar (ADD) se puede representar como:

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \text{campo-1} \\ \text{constante-1} \end{array} \right\} \underline{\text{TO}} \text{ campo-2}$$

El *campo-2* verá incrementado su valor con el valor de *campo-1* o en la *constante-1:*

$$\text{Suma} \left\{ \begin{array}{l} \text{campo-1} \\ \text{constante-1} \end{array} \right\} \text{a campo-2}$$

Es la forma más sencilla de la suma. La instrucción anterior se puede ver ampliada:

$$\underline{\text{ADD}} \left\{ \begin{matrix} \text{campo-1} \\ \text{constante-1} \end{matrix} \right\} \quad \left[\begin{matrix} \text{campo-2} \\ \text{constante-2} \end{matrix} \right] \ldots \underline{\text{TO}} \text{ campo-n.}$$

Sería equivalente a dos instrucciones consecutivas:

$$\underline{\text{ADD}} \left\{ \begin{matrix} \text{campo-1} \\ \text{constante-1} \end{matrix} \right\} \underline{\text{TO}} \text{ campo-n.}$$

$$\underline{\text{ADD}} \left\{ \begin{matrix} \text{campo-2} \\ \text{constante-2} \end{matrix} \right\} \underline{\text{TO}} \text{ campo-n.}$$

Otra variante de la instrucción sumar se presenta seguidamente:

$$\underline{\text{ADD}} \left\{ \begin{matrix} \text{campo-1} \\ \text{constante-1} \end{matrix} \right\} \quad \left[\begin{matrix} \text{campo-2} \\ \text{constante-2} \end{matrix} \right] \underline{\text{GIVING}} \text{ campo-n.}$$

En este caso *campo-n* toma como valor la suma de los dos operandos. Su significación podría ser:

$$\text{Suma} \left\{ \begin{matrix} \text{campo-1} \\ \text{constante-1} \end{matrix} \right\} \text{ y } \left\{ \begin{matrix} \text{campo-2} \\ \text{constante-2} \end{matrix} \right\}$$

y el resultado lo pone en *campo-n*.

En el siguiente programa se ilustra esta instrucción:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-SUMA.

*
* Este programa es un ejemplo del empleo de las instrucciones ADD.
*

 ENVIRONMENT DIVISION.

 DATA DIVISION.

 FILE SECTION.

 WORKING-STORAGE SECTION.

 01 CAMPO-1                  PIC 9(4)          VALUE ZERO.
 01 CAMPO-2                  PIC 9(4)          VALUE ZERO.
 01 CAMPO-3                  PIC 9(2).
 01 CAMPO-4                  PIC 9(4).
 01 CAMPO-5                  PIC 9(5).
 01 CAMPO-6                  PIC 9(3)          VALUE 1.

 01 MENSAJE.
    05 LITERAL               PIC X(33)         VALUE
        'EL RESULTADO DE LA OPERACION ES: '.
    05 CAMPO-RESULTADO       PIC 9(5).

 PROCEDURE DIVISION.
```

```
INICIO.
        DISPLAY 'TECLEE VALOR CAMPO-3: '.
        ACCEPT CAMPO-3.
        DISPLAY 'TECLEE VALOR CAMPO-4: '.
        ACCEPT CAMPO-4.
        DISPLAY 'TECLEE VALOR CAMPO-5: '.
        ACCEPT CAMPO-5.

PARTE-1.
        ADD CAMPO-3            TO CAMPO-4.
        MOVE CAMPO-4           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-2.
        MOVE CAMPO-2           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-1.
        MOVE CAMPO-1           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-2 CAMPO-5 TO CAMPO-6.
        MOVE CAMPO-6           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.

DAR-VALOR.
        DISPLAY 'TECLEE VALOR CAMPO-3: '.
        ACCEPT CAMPO-3.
        DISPLAY 'TECLEE VALOR CAMPO-4: '.
        ACCEPT CAMPO-4.
        DISPLAY 'TECLEE VALOR CAMPO-5: '.
        ACCEPT CAMPO-5.

PARTE-2.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-2.
        MOVE CAMPO-2           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-1.
        MOVE CAMPO-1           TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.

FINAL-PROGRAMA.
        STOP RUN.
```

Se trata de uno de los programas más largos realizados hasta el momento, pero es fácil de entender:

En la **WORKING-STORAGE SECTION** se han definido los campos que se van a emplear en el resto del programa: 6 campos numéricos y un campo denominado **MENSAJE**, que se subdivide en otros campos llamados **LITERAL** y **CAMPO-RESULTADO**.

La **PROCEDURE DIVISION** tiene 5 nombres de párrafo: uno de finalización (**FIN-PROGRAMA**), que contiene el **STOP RUN**; otros dos (**INICIO**, **DAR-VALORES**) se utilizan para leer de pantalla los valores iniciales de algunos campos. Los párrafos **PARTE-1** y **PARTE-2** contienen ejemplos de los diferentes tipos de **ADD's**.

Supóngase que mediante el juego de instrucciones ACCEPT de INICIO se dan los siguientes valores:

CAMPO-3			9	3
CAMPO-4	0	1	2	4
CAMPO-5	1	2	3	0

Después de la primera instrucción (ADD CAMPO-3 TO CAMPO-4), CAMPO-3 conservará su valor, mientras que CAMPO-4 será ahora

CAMPO-4	0	2	1	7

Y el mensaje que aparece en la pantalla:

El resultado de la operación es: 0217

Cuando se ejecuta ADD CAMPO-3 CAMPO-4 TO CAMPO-2, éste último tomará el valor:

CAMPO-3			9	3
CAMPO-4	0	2	1	7
CAMPO-2	0	3	1	0

En vez del valor 0 que tenía inicialmente (recordar que se le asignó mediante la cláusula VALUE ZERO, en la definición del campo realizada en la WORKING STORAGE.)

La expresión aritmética que se corresponde con la instrucción anterior es:

CAMPO-2 = CAMPO-2 + CAMPO-3 + CAMPO-4

Cuando se ejecuta la instrucción ADD CAMPO-3 CAMPO-4 TO CAMPO-1, como la fórmula es idéntica, el resultado coincide plenamente con el anterior.

Sin embargo, si el programa se modifica de la siguiente manera:

```
01 CAMPO-1            PIC 9(4).
```

eliminando la cláusula VALUE en la definición, al ejecutar de nuevo el programa se produce un error.

La razón es muy simple: CAMPO-1 no tiene valor inicial, no se le ha asignado ninguno y no se conoce su contenido. Es muy importante este «pequeño» detalle: es necesario dar un valor inicial a aquellas variables que lo necesiten.

En la siguiente instrucción también se presenta una peculiaridad: el campo que interviene en la suma como resultado tiene un PIC 9(3), mientras que el resultado (CAMPO-2 + CAMPO-5 + CAMPO-6 = CAMPO-6) (1541) tiene 4 cifras. Ya se recuerda que se comienza a ajustar por la derecha y, por tanto, la última cifra se perderá.

En pantalla se visualiza el mensaje:

El resultado de la operación es: 541.

Este tambien es un aspecto a tener en cuenta: es necesario asegurarse de que en las operaciones no se producirá un truncamiento que podría acarrear graves errores.

Siguiendo con la ejecución del programa, nuevamente se realiza la petición de valores para las variables CAMPO-3, CAMPO-4, CAMPO-5. Para poder comparar los resultados con los obtenidos anteriormente, los valores tecleados serán, respectivamente: 93, 124, 1230.

La instrucción ADD CAMPO-3 CAMPO-4 GIVING CAMPO-2 obtiene como resultado el valor 137 (93 + 124), independientemente de que el valor actual de CAMPO-2 era 310 (no se había modificado desde la última vez que se utilizó). En definitiva, se puede decir que responde a la fórmula: CAMPO-2 = CAMPO-3 + CAMPO-4. En este caso no es necesario dar un valor inicial a CAMPO-2, como se puede comprobar con la siguiente instrucción, cuyo resultado coincide plenamente con el anterior, a pesar de que el valor inicial de CAMPO-1 es distinto.

Como resumen y conclusión se puede decir que:

ADD... TO. Incrementa el valor inicial del campo resultado.

ADD... GIVING. Destruye el valor inicial, sustituyéndolo por la suma de los operandos.

En este punto es necesario hacer un inciso y exponer los campos numéricos reales, con cifras decimales.

La picture que corresponde a un campo de este tipo tiene la forma:

01 CAMPO-DECIMAL PIC 9(N)V9(D)

donde:

N: número de cifras enteras.
D: número de cifras decimales.
V: representa la coma decimal; no se almacena en el ordenador, sólo se emplea en operaciones aritméticas para alinear las distintas cifras.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-DECIMALES.

*
* Este programa es un ejemplo de la adición empleando
* variables no enteras.
*

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.
```

```
01 CAMPO-1                    PIC 9(3)V9(2).
01 CAMPO-2                    PIC 9(2)V9(3).
01 CAMPO-EDICION              PIC 9(3).9(2).

PROCEDURE DIVISION.

INICIO.
        MOVE 125.6      TO CAMPO-1.
        MOVE 23.923     TO CAMPO-2.
        ADD CAMPO-2 8.4 TO CAMPO-1.
        DISPLAY 'EL RESULTADO ES: ' CAMPO-1.
        DISPLAY '¡OBSERVAR QUE LA "V" NO '
                'TIENE REPRESENTACION EXTERNA!'.
        MOVE CAMPO-1    TO CAMPO-EDICION.
        DISPLAY 'EL RESULTADO AHORA ES: ' CAMPO-EDICION.
        DISPLAY 'PORQUE SE HA EMPLEADO EL PUNTO EN EDICION'

FIN-PROGRAMA.
        STOP RUN.
```

La primera curiosidad que surge en este programa es el empleo del punto decimal en lugar de la coma. Hay que recordar que se está utilizando notación anglosajona.

El formato de los campos y la manera de sumarlos se representan en la figura siguiente.

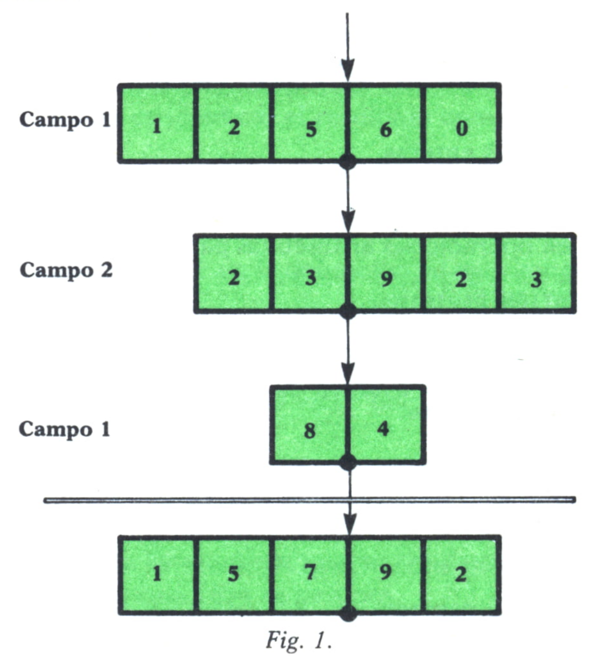


Fig. 1.

Se observa que a partir del punto decimal se ajusta a la izquierda y se rellena con ceros a la derecha. Igualmente se puede apreciar que los caracteres decimales también pueden sufrir truncamiento.

En el mensaje se displaya toda la cifra seguida, sin separación decimal, ya que la V sólo se emplea en los cálculos internos y no tiene representación externa.

Después del amplio tratamiento dado a la operación sumar, muchos conceptos que también se emplean en las siguientes instrucciones habrán quedado lo suficientemente claros, por lo que no se considera necesario abundar en ellos.

Sólo remarcar dos aspectos:

— Es necesario considerar qué variables deben tener un valor inicial y definir éste.

— Los truncamientos siempre son peligrosos, porque falsean los resultados.



RESTAR

La operación restar se representa mediante el vocablo inglés SUBTRACT.

Su formato más usual es:

$$\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \text{campo-1} \\ \text{const-1} \end{matrix} \right\} \underline{\text{FROM}} \text{ campo-2.}$$

Donde el valor de *campo-2* se ve decrementado en el valor de *campo-1* o *const-1:*

$$\text{campo-2} = \text{campo-2} - \left\{ \begin{matrix} \text{campo-1} \\ \text{const-1} \end{matrix} \right\}$$

(Atención con la inicialización de *campo-2*).

La serie de instrucciones:

$$\underline{\text{SUBTRACT}} \quad \left\{ \begin{matrix} \text{campo-1} \\ \text{const-1} \end{matrix} \right\} \underline{\text{FROM}} \text{ campo-n.}$$

$$\underline{\text{SUBTRACT}} \quad \left\{ \begin{matrix} \text{campo-2} \\ \text{const-2} \end{matrix} \right\} \underline{\text{FROM}} \text{ campo-n.}$$

Es equivalente a:

$$\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \text{campo-1} \\ \text{const-1} \end{matrix} \right\} \quad \left[\begin{matrix} \text{campo-2} \\ \text{const-2} \end{matrix} \right] \ldots \underline{\text{FROM}} \text{ campo-n.}$$

El único campo que ve modificado su valor es *campo-n.*

Si se emplea la instrucción de la forma:

$\underline{\text{SUBTRACT}} \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix} \underline{\text{FROM}} \begin{Bmatrix} \text{campo-2} \\ \text{const-2} \end{Bmatrix} \underline{\text{GIVING}}$ campo-3.

El valor de *campo-3* es «machacado» por el valor de la operación

$$\begin{Bmatrix} \text{campo-2} \\ \text{const-2} \end{Bmatrix} - \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix}$$

Ni *campo-1* ni *campo-2* ven modificado su valor.



MULTIPLICAR

Tiene como formatos:

$\underline{\text{MULTIPLY}} \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix} \underline{\text{BY}}$ campo-2.

$\underline{\text{MULTIPLY}} \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix} \text{campo-2} \\ \text{const-2} \end{Bmatrix} \underline{\text{GIVING}}$ campo-3.

que se corresponden con las siguientes fórmulas aritméticas:

$$\text{campo-2} = \text{campo-2} * \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix}$$

$$\text{campo-3} = \begin{Bmatrix} \text{campo-2} \\ \text{const-2} \end{Bmatrix} * \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix}$$

De donde se deduce que en el primer formato únicamente el multiplicando *(campo-1* o *const-1)* pueden ser un literal numérico. *Campo-2* ha de ser una variable, puesto que va a intervenir en la multiplicación, y contendrá el resultado de la misma. En el segundo formato, el valor inicial de *campo-3* es indiferente, y tanto el multiplicando como el multiplicador pueden ser constantes.



DIVIDIR

Es una instrucción de la forma:

$\underline{\text{DIVIDE}} \begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix} \underline{\text{INTO}}$ campo-2.

Donde: *campo-2* es el dividendo y contendrá el resultado de la división.

$\begin{Bmatrix} \text{campo-1} \\ \text{const-1} \end{Bmatrix}$ divisor, su valor permanece a lo largo de la división.

Así la serie de instrucciones:

MOVE 125 TO *campo-2*.

DIVIDE 5 INTO *campo-2*.
DISPLAY *campo-2*.

Visualizaría en la pantalla 25, que es el nuevo valor de *campo-2*.
Es preciso resaltar que la división por cero produce un error.
El siguiente formato utiliza la opción GIVING, que permite que tanto el dividendo como el divisor puedan ser constantes.

$$\underline{\text{DIVIDE}}\left\{\begin{array}{l}\text{campo-1}\\\text{const-1}\end{array}\right\}\ \underline{\text{INTO}}\left\{\begin{array}{l}\text{campo-2}\\\text{const-2}\end{array}\right\}\ \underline{\text{GIVING}}\ \text{campo-3}\ [\underline{\text{REMAINDER}}\ \text{campo-4}].$$

Se divide $\left\{\begin{array}{l}\textit{campo-2}\\\textit{const-2}\end{array}\right\}$ por $\left\{\begin{array}{l}\textit{campo-1}\\\textit{const-1}\end{array}\right\}$ llevando a *campo-3* el cociente. Si se elige la opción REMAINDER *campo-4*, en *campo-4* se almacenará el resto de la división entera.
También es posible realizar una división:

$$\underline{\text{DIVIDE}}\left\{\begin{array}{l}\text{campo-1}\\\text{const-1}\end{array}\right\}\ \underline{\text{BY}}\left\{\begin{array}{l}\text{campo-2}\\\text{const-2}\end{array}\right\}\ \underline{\text{GIVING}}\ \text{campo-3}\ [\text{REMAINDER...}].$$

En este caso $\left\{\begin{array}{l}\textit{campo-1}\\\textit{const-1}\end{array}\right\}$ es el dividendo, $\left\{\begin{array}{l}\textit{campo-2}\\\textit{const-2}\end{array}\right\}$ el divisor y en *campo-3* se almacenará el cociente.

MOVE 125 TO *campo-2*.
DIVIDE *campo-2* BY 5 GIVING *campo-3*.
DISPLAY *campo-3*.

Sería equivalente al ejemplo anterior.



COMPUTE

Operación que permite transferir a un campo el resultado de una expresión aritmética.

<u>COMPUTE</u> campo = EXPRESION-ARITMETICA.

Una expresión aritmética o fórmula matemática se compone de una serie de operadores que relacionan una serie de operandos (campos o constantes).
Los operadores empleados en COBOL son:

+ : Suma.
– : Resta.
* : Multiplicación.
/ : División.
** : Exponenciación.

Siendo opcional el uso de paréntesis.

Hay que tener en cuenta que cada operador debe ir precedido y seguido de un espacio en blanco.

La forma en que se calcula el resultado de una expresión aritmética sigue las reglas:

— En primer lugar, se ejecutan todas las exponenciaciones; si hubiese varias, serán prioritarias las que se encuentren más a la izquierda.

— A continuación se calculan las multiplicaciones (*) y divisiones (/), siempre de izquierda a derecha.

— Por último, se efectúan las operaciones de sumar (+) y restar (–).

Este orden puede ser modificado con el empleo de paréntesis, ya que éstos se evalúan con anterioridad y siguiendo las reglas anteriores.

El pequeño programa que se muestra a continuación sirve para aclarar todos estos aspectos.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-COMPUTE.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 CAMPO-1                PIC 9(2)         VALUE 12.
01 CAMPO-2                PIC 9(2)         VALUE  5.
01 CAMPO-3                PIC 9(2)         VALUE  4.
01 RESULTADO              PIC 9(2).

PROCEDURE DIVISION.

PRINCIPAL.
       COMPUTE RESULTADO = CAMPO-2 * CAMPO-3 + CAMPO-1.
       DISPLAY 'EL RESULTADO ES: ' RESULTADO.
       COMPUTE RESULTADO = CAMPO-2 * (CAMPO-3 + CAMPO-1).
       DISPLAY 'EL RESULTADO ES: ' RESULTADO.

FIN-PROGRAMA.
       STOP RUN.
```

Como puede comprobarse, el resultado de la primera operación se calcula de la siguiente forma:

— Primero, la multiplicación entre CAMPO-2 y CAMPO-3: (20).

— Segundo, la suma entre el resultado parcial obtenido hasta el momento y CAMPO-1: (20 + 12 = 32).

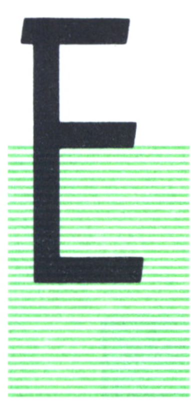
En la segunda operación el uso de paréntesis modifica el orden de ejecución.

Antes que nada se calcula el valor de la expresión encerrada entre paréntesis: (16) y a continuación se multiplica por CAMPO-2: (16 * 5 = 80).

Se ha podido observar que no es necesario iniciar el campo RESULTADO, ya que su valor es sustituido por el de la expresión aritmética.



SENTENCIAS CONDICIONALES



EN todos los ejemplos realizados hasta el momento, las instrucciones se ejecutaban en secuencia, es decir, que después de ejecutarse una instrucción, pasaba a realizarse la que le seguía.

Esta situación no es la que se presenta normalmente a un programador. Lo habitual es que se presenten alternativas, eligiéndose el camino adecuado en función de los datos introducidos.

Las alternativas en un programa COBOL se presentan con la siguiente sentencia:

```
IF condición
    sentencia ...
[ELSE sentencia...]
```

Como aplicación de esta sentencia, se realiza un ejemplo que resuelve el siguiente problema: se leerán dos datos numéricos; si el primero es mayor que el segundo, estos dos valores se restan. En caso contrario, se suman. En ambos casos se debe imprimir por pantalla el resultado.

En primer lugar, se muestra un organigrama que ayudará a realizar el programa.

Los dos primeros símbolos que aparecen en el organigrama son de lectura de los dos primeros datos. El siguiente es un rombo que representa una sentencia condicional. En ella se pregunta si el DATO1 es mayor que el DATO2.

Del rombo salen dos ramas, identificadas cada una de ellas por un «SI» o un «NO».

El flujo del organigrama irá por una u otra rama en función de la respuesta obtenida.

En cada una de las ramas se realiza una operación, escribiéndose en la siguiente sentencia el resultado.

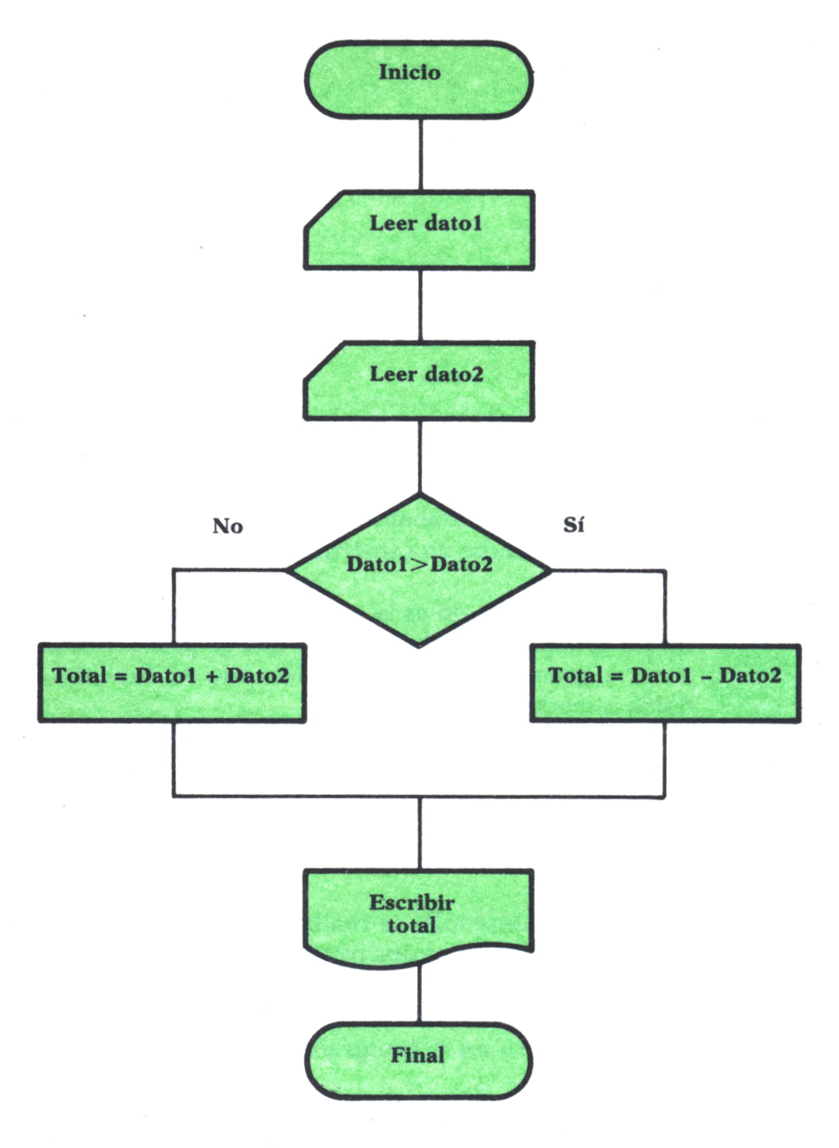


Fig. 1.

A continuación se realiza un seguimiento del organigrama.

Supóngase que los valores de DATO1 y DATO2 son 5 y 3, respectivamente.

La comparación será cierta (5 es mayor que 3), luego el flujo continuará por la rama del «SI». Se realiza la operación (5 – 3) y el resultado se imprimirá.

Si los datos anteriores se hubiesen introducido al revés (DATO1 con 3 y DATO2 con 5), la condición será falsa, yéndose el flujo por la rama del «NO»; procesándose la suma.

Cualquier sentencia condicional que aparezca en un programa COBOL sólo tiene dos posibles respuestas:

— SI o VERDAD.
— NO o FALSA.

El paso de este organigrama a un programa COBOL es el siguiente:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EJ-IF.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 DATO1                        PIC 9(3).
01 DATO2                        PIC 9(3).
01 TOTAL                        PIC 9(4).


PROCEDURE DIVISION.

INICIO.
         DISPLAY 'INTRODUZCA EL PRIMER DATO'.
         ACCEPT DATO1.
         DISPLAY 'INTRODUZCA EL SEGUNDO DATO'.
         ACCEPT DATO2.
         IF DATO1 > DATO2
             COMPUTE TOTAL = DATO1 - DATO2
         ELSE
             COMPUTE TOTAL = DATO1 + DATO2.
         DISPLAY 'EL RESULTADO ES: ' TOTAL.

FIN-PROGRAMA.
         STOP RUN.
```

Se toman los dos datos con las correspondientes sentencias ACCEPT's.

A continuación se encuentra la sentencia condicional. A la partícula IF le sigue la condición (más adelante se explicarán todos los tipos de condiciones).

La rama correspondiente al «SI» está formada por todas las instrucciones (en el ejemplo sólo hay una) hasta encontrar el ELSE, y la rama del «NO» comprende todas las instrucciones que siguen a la partícula ELSE hasta encontrar el primer punto.

Si la condición fuese cierta, se ejecutaría la instrucción COMPUTE TOTAL = DATO1 - DATO2, pasando a ejecutarse después la sentencia DISPLAY, es decir, saltando las sentencias que siguen al ELSE hasta el primer punto.

Al contrario pasaría si la condición fuese falsa.

En las instrucciones condicionales tiene especial importancia la situación del punto.

Si la instrucción COMPUTE que sigue al ELSE no finalizase en punto, la sentencia que sigue a la primera sería parte de la rama del «NO», rompiéndose la lógica del programa.

Este es un ejemplo que muestra lo peligroso que puede ser la mala colocación de un punto.

Una condicional no es necesario que contemple los dos caminos, en muchas ocasiones no se utiliza la cláusula ELSE. En estos casos, si la condición es cierta se ejecutarán todas las instrucciones que siguen al IF hasta encontrar el primer punto (ya que no está el ELSE).

El formato de las condiciones es el siguiente:

$$\left\{\begin{array}{l}\text{campo-1}\\ \text{literal-1}\end{array}\right\}\ \underline{\text{IS}}\ [\text{NOT}]\ \left\{\begin{array}{c} > \\ < \\ = \\ \text{GREATER THAN} \\ \text{LESS THAN} \\ \text{EQUAL}\end{array}\right\}\ \left\{\begin{array}{l}\text{campo-2}\\ \text{literal-2}\end{array}\right\}.$$

Como se deduce del formato, se puede comparar un campo con otro o con un literal. La única combinación prohibida es la de comparar dos literales, ya que sería absurdo, porque el resultado será conocido «a priori».

Las condiciones que se pueden presentar son:

— > o GREATER THAN	Mayor que.
— < o LESS THAN	Menor que.
— = o EQUAL	Igual.

El COBOL permite utilizar símbolos de comparación o su correspondiente en letras.

Para contemplar todas las posibles condiciones que se puedan presentar se utiliza la partícula NOT.

— NOT > o NOT GREATER THAN	Menor o igual que.
— NOT < o NOT LESS THAN	Mayor o igual que.
— NOT = o NOT EQUAL	Distinto que.

El siguiente programa, lee tres datos y los imprime en orden decreciente.

El organigrama que lo contempla es:

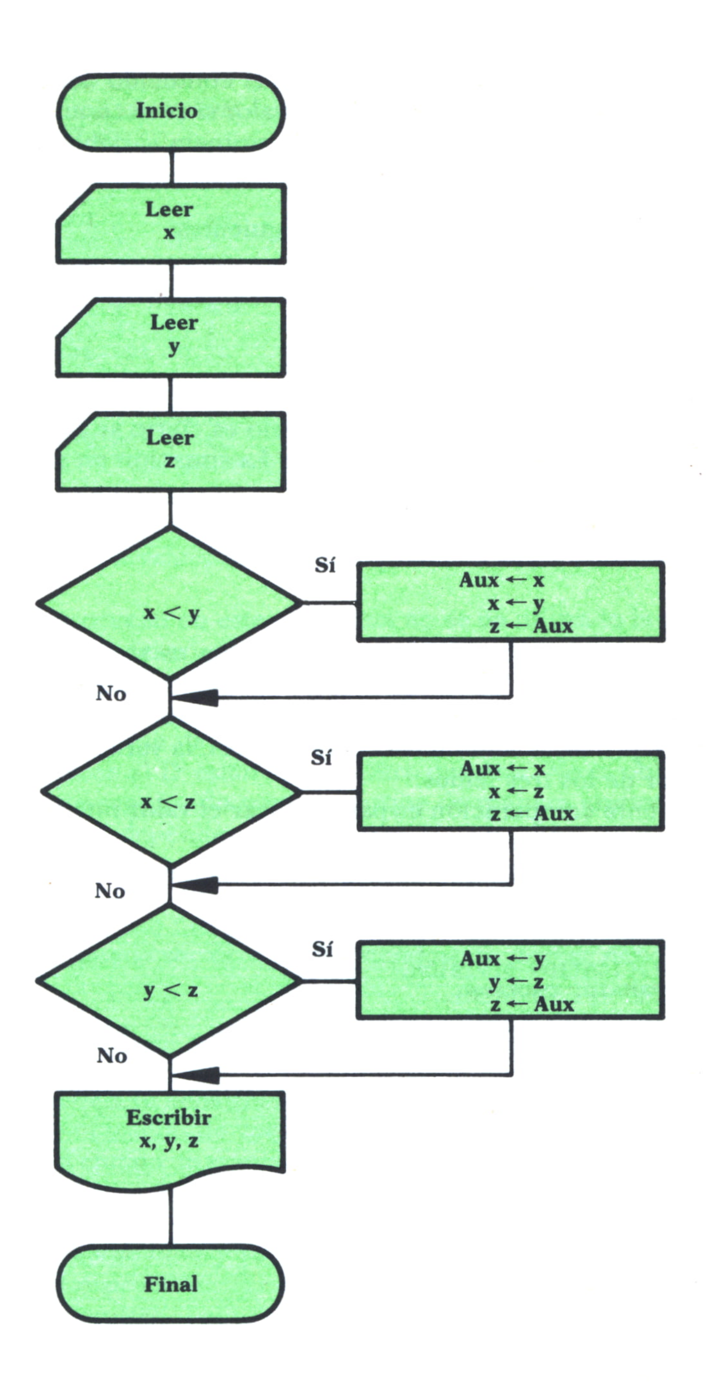

Inicio
Leer
x
Leer
y
Leer
z
x < y
Sí
Aux ← x
x ← y
z ← Aux
No
x < z
Sí
Aux ← x
x ← z
z ← Aux
No
y < z
Sí
Aux ← y
y ← z
z ← Aux
No
Escribir
x, y, z
Final

Fig. 2.

El proceso consiste en ir comparando el contenido del primer campo con todos los demás, intercambiando su valor si es necesario, de tal forma que siempre quede el valor más alto en la primera variable, y así sucesivamente.

Si los datos introducidos fuesen los siguientes:

X ← 6
Y ← 9
Z ← 9

El proceso sería:

Se compara si X es menor que Y, como es cierto se intercambian sus valores (ayudándose de la variable AUX). La situación en ese punto es:

X ← 9
Y ← 6
Z ← 9

A continuación la condición que se evalúa es si X es menor que Z, como es falsa (9 no es mayor que 9), no se ejecuta ninguna instrucción.

En este momento, fuesen cuales fuesen los valores de entrada, después de estas dos primeras sentencias condicionales, la variable X contendrá el valor mayor de los tres leídos.

La siguiente comparación es para colocar el valor intermedio en la variable Y y el menor en Z.

El programa COBOL que corresponde al anterior organigrama es:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ORDENAR.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 X              PIC 99.
01 Y              PIC 99.
01 Z              PIC 99.
01 AUX            PIC 99.

PROCEDURE DIVISION.

INICIO.
      DISPLAY 'INTRODUZCA PRIMER VALOR'.
      ACCEPT X.
```

```
          DISPLAY 'INTRODUZCA SEGUNDO VALOR'.
          ACCEPT Y.
          DISPLAY 'INTRODUZCA TERCER VALOR'.
          ACCEPT Z.
          IF X < Y
             MOVE X   TO AUX
             MOVE Y   TO X
             MOVE AUX TO Y.
          IF X < Z
             MOVE X   TO AUX
             MOVE Z   TO X
             MOVE AUX TO Z.
          IF Y < Z
             MOVE Y   TO AUX
             MOVE Z   TO Y
             MOVE AUX TO Z.
          DISPLAY 'LOS VALORES ORDENADOS SON: ' X ' ' Y ' ' Z.

      FIN-PROGRAMA.
          STOP RUN.
```

En los IF que aparecen en este programa no se ha utilizado la cláusula ELSE porque no es necesario, finalizando los IF en el primer punto que encuentra.

Si se observa la forma de escribir el programa, se ve que las instrucciones que pertenecen al IF están más adentro que éste. No es obligatorio, pero sí conveniente seguir esta norma, porque de manera gráfica se plasma qué instrucciones dependen de qué otras.

En el formato del IF se ha visto que dentro del cuerpo va un conjunto de instrucciones, y se han mostrado algunos ejemplos de las mismas. Una instrucción válida puede ser otro IF, formándose lo que se denomina IF anidados.

Se explicará esta nueva estructura sobre un ejemplo.

Una empresa pide a su departamento de informática que realice un programa para hallar el sueldo de cada empleado. Para ello, se toman como datos de entrada el código de empleado, sueldo, número de hijos y antigüedad.

Si el empleado tiene hijos y su sueldo supera las 100.000 ptas., su gratificación será del 6 por 100 . En cambio, si su retribución es inferior o igual, será del 10 por 100.

Si no tiene hijos, pero su antigüedad es mayor de 7 años, la gratificación es del 6 por 100, para antigüedad inferior, será del 3 por 100.

El organigrama que aborda el problema es el siguiente:

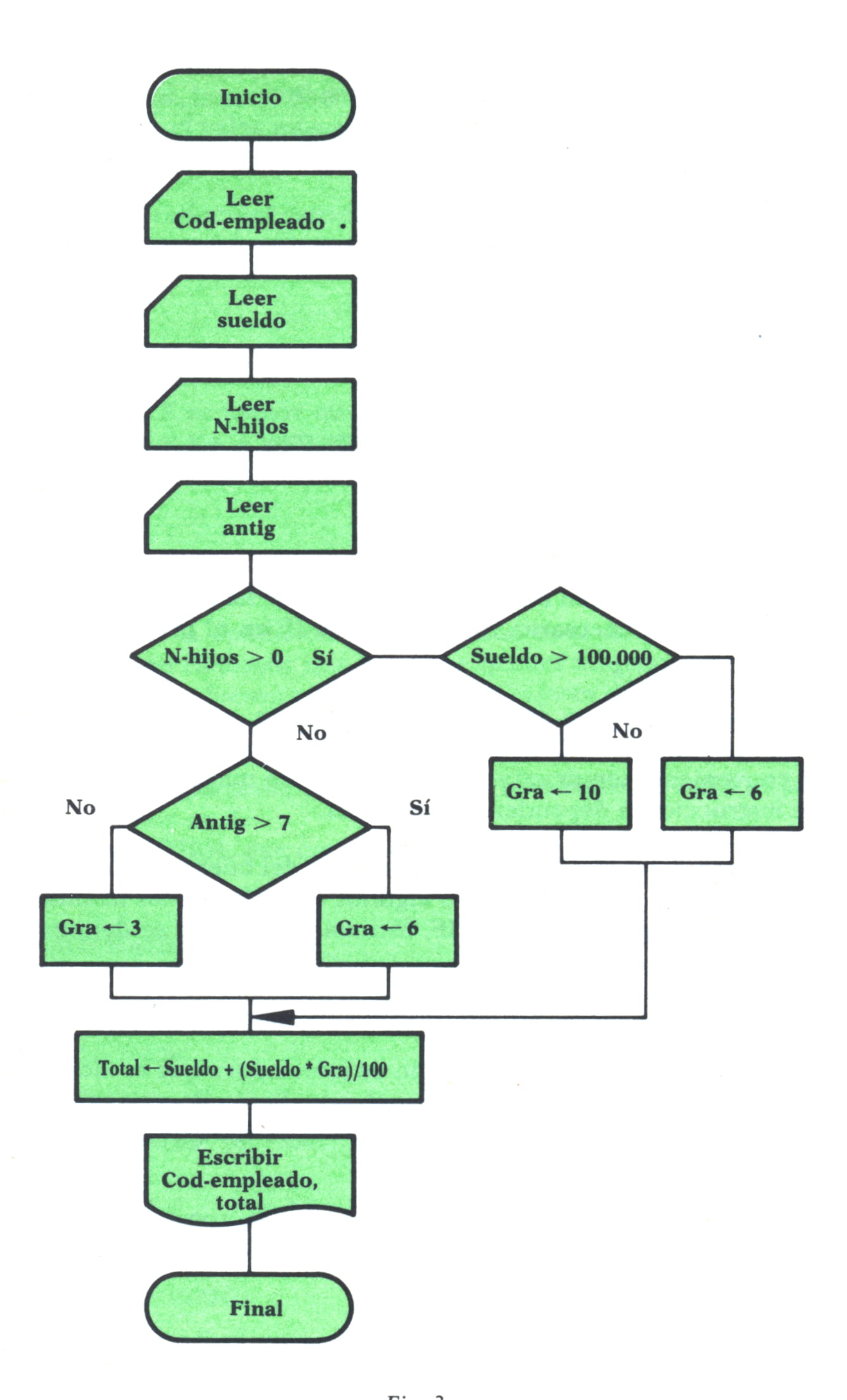

Inicio
Leer
Cod-empleado
Leer
sueldo
Leer
N-hijos
Leer
antig
N-hijos > 0
Sí
No
Sueldo > 100.000
No
Gra ← 10
Gra ← 6
No
Antig > 7
Sí
Gra ← 3
Gra ← 6
Total ← Sueldo + (Sueldo * Gra)/100
Escribir
Cod-empleado,
total
Final

Fig. 3.

Como se observa en esta figura, existen condiciones incluidas dentro de otras.

El paso a COBOL genera el siguiente programa:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EMPLEADO.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 FICHA-EMPLEADO.
   05 COD-EMPLEADO                    PIC X(5).
   05 SUELDO                          PIC 9(6).
   05 N-HIJOS                         PIC 99.
   05 ANTIG                           PIC 99.

01 TOTAL                              PIC 9(7).
01 GRATI                              PIC 99.

PROCEDURE DIVISION.

INICIO.
       DISPLAY 'CODIGO EMPLEADO'.
       ACCEPT COD-EMPLEADO.
       DISPLAY 'SUELDO'.
       ACCEPT SUELDO.
       DISPLAY 'NUMERO DE HIJOS'.
       ACCEPT N-HIJOS.
       DISPLAY 'ANTIGUEDAD'.
       ACCEPT ANTIG.
       IF N-HIJOS > 0
          IF SUELDO > 100000
             MOVE 6  TO GRATI
          ELSE
             MOVE 10 TO GRATI
       ELSE
          IF ANTIG > 7
             MOVE  6 TO GRATI
          ELSE
             MOVE  3 TO GRATI.
       COMPUTE TOTAL ROUNDED = SUELDO + (SUELDO * GRATI) / 100.
       DISPLAY COD-EMPLEADO ' DEBE RECIBIR ' TOTAL ' Pts. '.
*
* La opción ROUNDED redondea los valóres decimales que se hayan
* podido obtener.
*

FIN-PROGRAMA.
       STOP RUN.
```

El funcionamiento de los IF's anidados es similar al de los IF's simples. La única consideración que hay que realizar es la correspondencia entre el IF y el ELSE.

La regla que hay que seguir es:

El primer ELSE que se encuentre en la estructura corresponde al primer IF que le preceda y que no esté asignado ya.

El primer ELSE del ejemplo se corresponde con IF SUELDO > 100000, que es el primer IF libre que le precede. El segundo se relaciona con IF N-HIJOS > 04, que es el primer IF libre anterior, ya que el que le precede inmediatamente ya está asignado.

De esta forma se evita cualquier ambigüedad que se pudiera producir.

En el ejemplo, gráficamente se pueden adivinar las dependencias y relaciones entre IF's y ELSE's. Un IF está relacionado con el ELSE que esté a su misma altura y depende de un IF que lo incluya. Es un nuevo detalle que resalta la importancia del "estilo" de escribir programas.

Volviendo al ejemplo anterior, la dirección de la empresa decide suspender la prima del 10 por 100; por tanto, hay que modificar el programa.

El primer impulso consiste en quitar el primer ELSE y la instrucción que le sigue, quedando la estructura como sigue:

```
IF N-HIJOS > 0
   IF SUELDO > 100000
      MOVE 6 TO GRA

ELSE
   IF ANTIG > 7
      MOVE 6 TO GRA

   ELSE
      MOVE 3 TO GRA.
```

Esta solución es errónea. Si se aplica la regla antes desarrollada, el primer ELSE se relaciona con IF SUELDO > 100000, no con IF N-HIJOS > 0, porque es el primer IF que queda libre.

Para solventar estas situaciones, se utiliza la sentencia NEXT SENTENCE. Esta instrucción no realiza ningún tipo de operación, simplemente da paso a la siguiente instrucción que se deba ejecutar.

Considerando esta nueva sentencia, la modificación sería la siguiente:

```
IF N-HIJOS > 0
   IF SUELDO > 100.000
      MOVE 6 TO GRA

   ELSE
          NEXT SENTENCE
```

```
ELSE
   IF ANTIG > 7
      MOVE 6 TO GRA
   ELSE
     MOVE 3 TO GRA.
```

Se ha introducido un ELSE que no realiza ninguna operación, pero que permite mantener la estructura de IF's anidados.

El COBOL permite un nivel de anidamiento de IF's ilimitado, aunque los diferentes compiladores lo acotan a un número fijo.

Todas las condiciones que se han visto hasta el momento son simples, pero también pueden formarse condiciones compuestas utilizando los operadores lógicos.

Existen dos operadores lógicos:

— AND
— OR

El funcionamiento de ambos operadores se muestra en la siguiente tabla de verdad:

C1	C2	C1 and C2	C1 or C2
F	F	F	F
V	F	F	V
F	V	F	V
V	V	V	V

C1: Condición.
C2: Condición.

F: Falso.
V: Verdad.

Fig. 4.

Se observa que al aplicarse el operador AND sobre una condición que sea falsa, el resultado ya es falso. Deben ser las dos verdaderas para que lo sea la condición compuesta.

En cambio, en una condición calificada por un OR, basta con que sea cierta al menos una de las dos condiciones para que sea verdad el conjunto.

Estos conceptos se explican con el siguiente ejemplo:

En función del valor de tres códigos, debe asignarse determinados literales, como indica la siguiente tabla:

Códigos	Literales
A = 1 y B = 1 y C = 1	APARTADO
(A = 1 y B = 1) o C > 3	REGISTRADO
A <> 1 y (B > 2 o C = 2)	ANOTADO
A > 1 o B = 0 o C = 0	TOTAL

El programa que refleja esta tabla es:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONDICIONALES.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 CODIGOS.
   05 A            PIC 9.
   05 B            PIC 9.
   05 C            PIC 9.

01 LITERAL         PIC X(10).


PROCEDURE DIVISION.

INICIO.
         DISPLAY 'CODIGO A'.
         ACCEPT A.
         DISPLAY 'CODIGO B'.
         ACCEPT B.
         DISPLAY 'CODIGO C'.
         ACCEPT C.
         IF A = 1 AND B = 1 AND C = 1
            MOVE 'APARTADO'    TO LITERAL.
         IF (A = 1 AND B = 1) OR C > 3
            MOVE 'REGISTRADO' TO LITERAL.
         IF A NOT = 1 AND ( B > 2 OR C = 2 )
            MOVE 'ANOTADO'     TO LITERAL.
         IF A > 1 OR B = 0 OR C = 0
            MOVE 'TOTAL'       TO LITERAL.
         DISPLAY LITERAL.

FIN-PROGRAMA.
         STOP RUN.
```

Todas las condiciones son triples y enlazadas con los dos operadores lógicos.

En las condiciones compuestas existe una prioridad de operadores. Primero se ejecutan todas las instrucciones relacionadas por un AND y después los OR.

Estas prioridades pueden ser rotas por los paréntesis. Lo que está encerrado entre ellos es lo primero que se evalúa.

La primera condición del programa está formada por los dos AND.

En la segunda se ejecutarían primero las dos condiciones unidas por el AND, y el resultado de ésta se relacionará mediante un OR con la tercera condición.

En la tercera, como se quiere realizar en primer lugar el OR, se encierra entre paréntesis, relacionándolo luego con el AND.

PERFORM 12



QUE ocurre cuando en un programa es necesario efectuar varias veces las mismas operaciones? ¿Hay que repetir una y otra vez dentro de la **PROCEDURE** las líneas que realizan esa operación?

La respuesta a estas preguntas es una nueva instrucción: **PERFORM**, que con sus muchos y variados formatos convierte al **COBOL** en un lenguaje estructurado y le proporciona una gran potencia.

Un ejemplo claro de la necesidad de repetir sentencias se encuentra en el programa explicado anteriormente (cap. 10, fig. 1), en el que los párrafos **INICIO** y **DAR-VALOR** son idénticos.

El formato más simple de esta sentencia es:

PERFORM nombre-de-párrafo.

Y lo que hace es ejecutar, en el lugar del programa en que se encuentre, todas aquellas instrucciones contenidas en el párrafo mencionado.

La estructura del programa de la fig. 1 del cap. 10 puede ser modificada y el resultado sería el mismo; con la ventaja adicional de tener un programa muy estructurado y fácilmente legible.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-SUMA.

*
* Este programa es un ejemplo del empleo de las instrucciones ADD.
*

 ENVIRONMENT DIVISION.

 DATA DIVISION.

 FILE SECTION.
```

```
WORKING-STORAGE SECTION.

01 CAMPO-1                  PIC 9(4)        VALUE ZERO.
01 CAMPO-2                  PIC 9(4)        VALUE ZERO.
01 CAMPO-3                  PIC 9(2).
01 CAMPO-4                  PIC 9(4).
01 CAMPO-5                  PIC 9(5).
01 CAMPO-6                  PIC 9(3)        VALUE 1.


01 MENSAJE.
   05 FILLER                PIC X(33)       VALUE
       'EL RESULTADO DE LA OPERACION ES: '.
   05 CAMPO-RESULTADO       PIC 9(5).


PROCEDURE DIVISION.

PRINCIPAL.
        PERFORM DAR-VALORES.
        PERFORM PARTE-1.
        PERFORM DAR-VALORES.
        PERFORM PARTE-2.
        STOP RUN.


DAR-VALORES.
        DISPLAY 'TECLEE VALOR CAMPO-3: '.
        ACCEPT CAMPO-3.
        DISPLAY 'TECLEE VALOR CAMPO-4: '.
        ACCEPT CAMPO-4.
        DISPLAY 'TECLEE VALOR CAMPO-5: '.
        ACCEPT CAMPO-5.


PARTE-1.
        ADD CAMPO-3             TO CAMPO-4.
        MOVE CAMPO-4            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-2.
        MOVE CAMPO-2            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-1.
        MOVE CAMPO-1            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-2 CAMPO-5 TO CAMPO-6.
        MOVE CAMPO-6            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.


PARTE-2.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-2.
        MOVE CAMPO-2            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
        ADD CAMPO-3 CAMPO-4 TO CAMPO-1.
        MOVE CAMPO-1            TO CAMPO-RESULTADO.
        DISPLAY MENSAJE.
```

Con respecto al programa anterior, se observa que los párrafos INICIO y DAR-VALOR se han fundido en uno solo: DAR-VALORES, que se ejecuta dos veces, debido a dos instrucciones PERFORM DAR-VALORES.

Pero el PERFORM tiene otra utilidad: permite repetir muchas veces una serie de sentencias al tiempo que controla el número de veces que se repite.

Imaginemos un pequeño programa que lee de pantalla la nota de todos los alumnos de una clase y calcula la nota media. Si los alumnos de la clase son 40, ¿hay que repetir 40 veces un PERFORM?

```
PROCEDURE DIVISION.

PRINCIPAL.
          MOVE ZERO TO SUMA.
          PERFORM LEER-NOTA.
          PERFORM LEER-NOTA.
                     .
                     .     40 VECES
                     .
          PERFORM LEER-NOTA.
          DIVIDE SUMA BY 40 GIVING MEDIA.
          DISPLAY MEDIA.
          STOP RUN.

LEER-NOTA.
          DISPLAY 'TECLEE NOTA: '.
          ACCEPT NOTA.
          ADD NOTA TO SUMA.
```

A simple vista se observa lo ridículo e ineficaz que resulta un programa de este tipo. Además, ¿qué ocurriría si la clase no tiene 40 alumnos? Habría que modificar y compilar nuevamente el programa.

Para solventar estas situaciones el COBOL dispone de una sentencia PERFORM que controla la ejecución de un párrafo en función de una condición que se evalúa previamente.

PERFORM nombre-de-párrafo UNTIL condición.

Su organigrama aparece en la siguiente figura

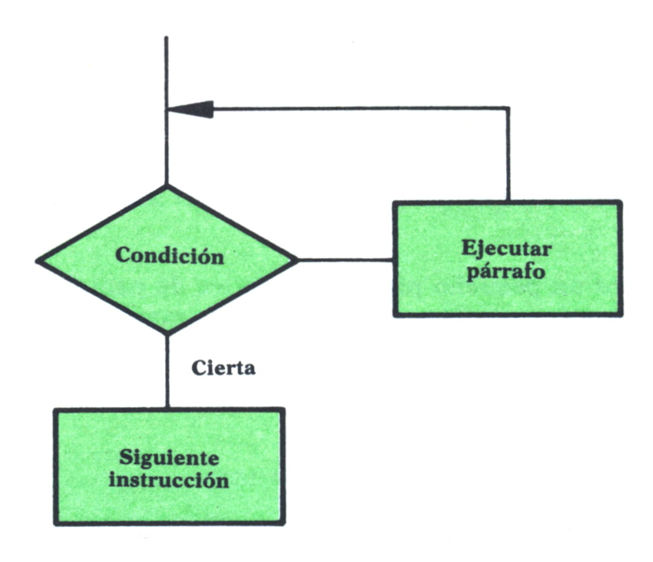


Fig. 3.

Antes de ejecutar las instrucciones que componen el párrafo se comprueba la condición: si es cierta, no se ejecuta el párrafo y se ejecuta la instrucción que se encuentra a continuación del PERFORM. Si no se cumple la condición, el párrafo se ejecuta una y otra vez, hasta que ésta se cumple.

En resumen, el significado de esta instrucción podría ser: ejecuta el párrafo «X» hasta que se cumpla la condición.

Si la condición resulta ser cierta ya la primera vez, el párrafo no se ejecutará nunca.

Veamos cómo queda el programa anterior:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NOTA-MEDIA.

ENVIRONMENT DIVISION.

DATA DIVISION.

FILE SECTION.

WORKING-STORAGE SECTION.

01 NUMERO-ALUMNOS          PIC 99.
01 CONTADOR                PIC 99          VALUE ZERO.
01 SUMA                    PIC 9(3)        VALUE ZERO.
01 MEDIA                   PIC 9(3).99.
01 NOTA                    PIC 99.
01 CORRECTA                PIC X.
```

```
PROCEDURE DIVISION.

PRÍNCIPAL.
        PERFORM LEER-ALUMNOS.
        PERFORM LEER-NOTA UNTIL CONTADOR = NUMERO-ALUMNOS.
        PERFORM DISPLAYAR-MEDIA.
        STOP RUN.

LEER-ALUMNOS.
        DISPLAY 'TECLEE NUMERO DE ALUMNOS '.
        ACCEPT NUMERO-ALUMNOS.

LEER-NOTA.
        MOVE 'N' TO CORRECTA.
        PERFORM LEER UNTIL CORRECTA = 'S'.
        ADD NOTA TO SUMA.
        ADD 1     TO CONTADOR.

LEER.
        DISPLAY 'TECLEE UNA NUEVA NOTA: '.
        ACCEPT NOTA.
        IF NOTA < 0 OR NOTA > 10
           DISPLAY '¡ERROR: NOTA INCORRECTA!'
        ELSE
           MOVE 'S' TO CORRECTA.

DISPLAYAR-MEDIA.
        DIVIDE SUMA BY NUMERO-ALUMNOS GIVING MEDIA.
        DISPLAY 'LA NOTA MEDIÁ DE LOS ' NUMERO-ALUMNOS
                ' ALUMNOS ES ' MEDIA.
```

El programa principal consta de 4 PERFORM's:

— El primero, LEER-ALUMNOS, tiene como única misión leer de pantalla el número de alumnos presentes en clase.

— A continuación el PERFORM LEER-NOTA hará que se repita este párrafo hasta que el número de veces que se hace esta lectura sea igual al número de alumnos que haya.

Para el control de la repetición se emplea la variable CONTADOR, que inicialmente tomará valor cero y se irá incrementando en uno por cada nueva lectura.

— Además, se realiza una validación de la nota tecleada: para que se considere correcta ha de ser mayor o igual que cero (NOTA $\geq$ 0) y menor o igual que 10 (NOTA $\leq$ 10). Mientras estas condiciones no se cumplan, se continuará pidiendo la misma nota (PERFORM LEER UNTIL CORRECTA).

— Cuando se hayan leído las notas de todos los alumnos (CONTADOR= = NUMERO-ALUMNOS), se calculará e imprimirá la MEDIA.

El organigrama de este programa puede ayudar a comprender y profundizar en el mismo; se representa en la figura siguiente:

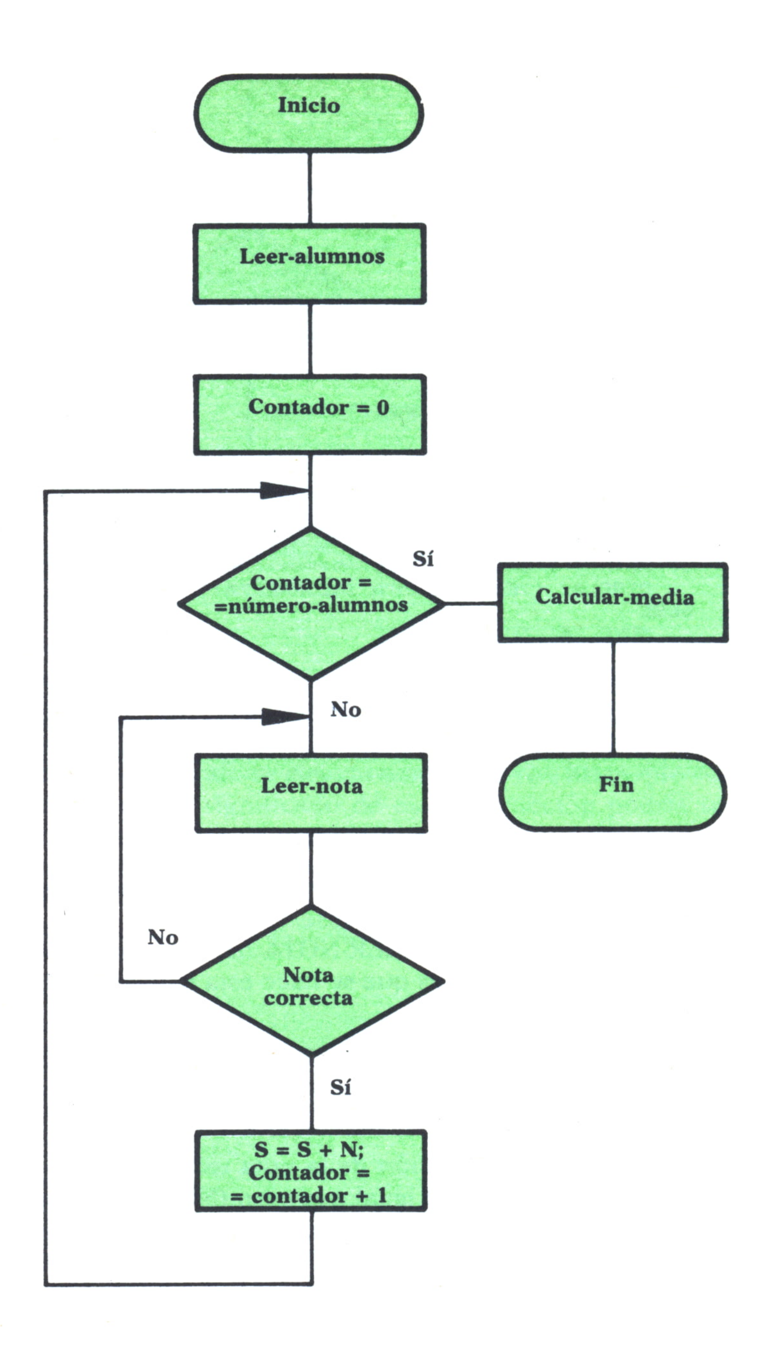
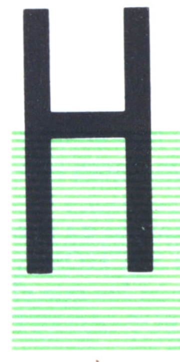

Inicio
Leer-alumnos
Contador = 0
Contador =
=número-alumnos
Sí
Calcular-media
No
Leer-nota
Fin
No
Nota
correcta
Sí
S = S + N;
Contador =
= contador + 1

Fig. 2.

Es preciso detenerse aquí y reflexionar: si todos los conceptos que se han explicado hasta el momento, en especial el IF y el PERFORM, se comprenden plenamente, el lector puede continuar el estudio del lenguaje COBOL.

Si no es así, quizá sea conveniente retomar la lectura en aquel punto donde surgieron las primeras dificultades, porque todas las sentencias expuestas se emplearán mucho en sucesivos apartados y su desconocimiento puede causar problemas en la comprensión de los siguientes ejemplos.

FICHEROS 13



ASTA el momento, y para la toma de datos, se ha empleado únicamente la pantalla, a través de instrucciones como DISPLAY y ACCEPT.

De igual manera se ha omitido la descripción de una de las divisiones del COBOL: ENVIRONMENT DIVISION, y de la sección FILE SECTION perteneciente a la DATA DIVISION.

Y es que, además de los periféricos estándar de entrada/salida: el teclado y la pantalla, existen otros tipos de dispositivos que realizan la entrada/salida de los programas de usuario: discos, cintas, impresoras...

Todos ellos deben declararse en COBOL y esa declaración se hace en la ENVIRONMENT.

Pero antes de explicar las sentencias de definición de ficheros es necesario conocer conceptos como organización y método de acceso.

¿Qué es un fichero de datos? Un conjunto de informaciones almacenadas según una determinada estructura lógica, denominada registro.

La forma en que esos registros se graban en el dispositivo se llama organización y puede ser secuencial o por clave.

— Secuencial: los registros se encuentran grabados en el mismo orden en que se han introducido, sea cual sea su valor.

— Por clave: existen unos campos dentro del registro que se emplean para ordenar el fichero. Los registros se graban en un lugar o en otro, dependiendo de los valores de estos campos. Estos campos se denominan clave del registro.

Se podría decir que la palabra organización se corresponde con el modo de «creación» del fichero.

Una vez creado un fichero es necesario acceder a él para tratar la in-

formación que contiene. La forma de acceso, llamada también método de acceso, es función de la organización y puede ser:

— Acceso secuencial: los registros se tratan uno a continuación de otro: para acceder al registro que está en la posición 5 es necesario pasar por los cuatro anteriores.

Se pueden recorrer de forma secuencial tanto los ficheros creados secuencialmente como los creados por clave.

Son ficheros secuenciales todos los almacenados en una cinta magnética.

— Acceso directo: se puede acceder a un registro en particular sin necesidad de recorrer los que le preceden.

El acceso directo es típico de ficheros cuyos registros se han almacenado dependiendo de su clave; suelen encontrarse grabados en discos (diskettes o discos duros).

La división ENVIRONMENT se emplea para definir el «entorno» en que se va a procesar el programa: ordenador (CONFIGURATION SECTION) y dispositivos en que se encuentran almacenados los ficheros con los que el programa va a trabajar; ya sea para lectura y/o escritura (INPUT-OUTPUT SECTION).

Su estructura es:



ENVIRONMENT DIVISION

```
[CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.]
[INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.]
```

Las dos secciones son opcionales, sólo son obligatorias cuando se utilizan ficheros.

Como es lógico suponer, los párrafos ENVIRONMENT, CONFIGURATION e INPUT-OUTPUT deben empezar en el margen A.



CONFIGURATION SECTION

Su principal misión es la de especificar en qué modelo de ordenador se ha desarrollado el programa:

SOURCE-COMPUTER. Describe el modelo de ordenador en que se ha escrito el programa fuente.

OBJECT-COMPUTER. Modelo en que se ejecuta el programa.

Los dos párrafos comienzan en el margen A.

En algunos compiladores estos dos párrafos no son necesarios, pero resulta conveniente ponerlos siempre, a efectos de legibilidad y mejor comprensión del programa.



SPECIAL-NAMES

Se emplea para intercambiar la misión de la coma y el punto decimal y su formato es:

DECIMAL-POINT IS COMMA.

Debe empezar en el margen B; si se omite la opción DECIMAL-POINT, se empleará la puntuación sajona: un punto entre las cifras enteras y decimales y una coma separando los millones; justo al revés de la notación castellana.



INPUT-OUTPUT SECTION

De los dos párrafos que componen esta sección sólo se va a explicar el que se llama FILE CONTROL, ya que la I-O-CONTROL apenas se utiliza.



FILE-CONTROL

En él se definen todos y cada uno de los ficheros que van a intervenir en el programa, describiendo el tipo de dispositivo en que se almacenan, su organización y el método de acceso. Todo esto se hace mediante una cláusula SELECT.

Hay que resaltar aquí que debido a las diferencias existentes entre los distintos tipos de ordenadores, la cláusula SELECT es la que más diferencias presenta en un modelo y otro. Por tanto, cuando un programa que está funcionando en un ordenador quiere migrarse a otro, es conveniente revisar dicha cláusula.

En este apartado se explica esta cláusula de acuerdo con el COBOL desarrollado por MICROSOFT para un IBM-PC (y compatibles).

{ SELECT nombre-de-fichero
ASSIGN TO dispositivo

ORGANIZATION IS { SEQUENTIAL / RANDOM }

```
                              RANDOM
[ACCESS METHOD IS  { SEQUENTIAL  }]
                              DYNAMIC
```

[RECORD KEY IS campo].

Todos estos párrafos han de empezar en el margen B, excepto la sentencia FILE-CONTROL, que lo hace en el margen A.

Se explican, paso a paso, todos los párrafos:

— Nombre de fichero. Se trata de un nombre de 8 caracteres como máximo, que debe empezar con una letra. Es el nombre que utiliza el programa para referirse al fichero.

— ASSIGN. Sirve para relacionar el fichero con el dispositivo en el que se encuentra almacenado. Los dispositivos más usuales son disco e impresora:

— ASSIGN TO DISK
— ASSIGN TO PRINTER, respectivamente.

— ORGANIZATION. Describe la forma en que el fichero ha sido o va a ser creado:

— SEQUENTIAL. Se corresponde con la organización secuencial.
— RANDOM. Organización por clave.

En esta versión del lenguaje COBOL existe otra organización:

— LINE SEQUENTIAL. Que reconoce el carácter de final de línea.
— ACCESS METHOD. Describe la forma de acceso al fichero:

— Secuencial: SEQUENTIAL
— Directa: RANDOM

La forma DYNAMIC indica que el fichero, dentro del mismo programa, va a ser accedido de forma secuencial y directa.

— RECORD KEY. Si se trata de un acceso directo, se utiliza esta cláusula para definir los campos que se emplearán como claves.

Como ejemplo de la definición de ficheros, tenemos el programa que a continuación se expone, en él se han utilizado dos ficheros:

El primero, ENTRADA, es un fichero secuencial que va a ser tratado de forma secuencial.

El otro, SALIDA, es un listado y está asociado con una impresora.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-SELECT.


ENVIRONMENT DIVISION.
```

```
CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PC.
   OBJECT-COMPUTER. IBM-PC.
   SPECIAL-NAMES.
         DECIMAL-POINT IS COMMA.

INPUT-OUTPUT SECTION.

   FILE-CONTROL.

         SELECT ENTRADA ASSIGN TO DISK
                        ORGANIZATION SEQUENTIAL
                        ACCESS SEQUENTIAL.

         SELECT SALIDA  ASSIGN TO PRINTER
                        ORGANIZATION SEQUENTIAL
                        ACCESS SEQUENTIAL.
```

Resumiendo, en la cláusula SELECT se hace la descripción, desde el punto de vista físico, de todos los ficheros intervinientes en el programa. Hay una sentencia SELECT para cada uno.

Además de los datos ya proporcionados del dispositivo, organización y método de acceso, es preciso definir la estructura del fichero: qué campos componen el registro, cuál es su longitud, si los registros se agrupan o no en bloques, etc.

Toda esta información se detalla en una sección de la DATA DIVISION llamada FILE SECTION.



FILE SECTION.

Este párrafo, cuando existe, empieza en el margen A y debe seguir a la sentencia DATA DIVISION.

Para cada uno de los ficheros utilizados se abre una cláusula FD (file description, descripción del fichero), que comienza en el margen A y va seguida por el nombre del fichero en cuestión. Su estructura es así:

FD nombre-de-fichero
LABEL RECORDS
[BLOCK CONTAINS]
[RECORDING MODE]
[DATA RECORDS]
01 nombre-del-registro.

Aunque algunos tipos de ordenadores emplean compiladores para los que es necesario especificar en el programa todas las cláusulas, la mayo-

ría de los compiladores prescinden del RECORDING MODE y DATA RECORDS, que pasan a ser meros comentarios.

Todas son opcionales, excepto la cláusula en que se definen las etiquetas que tiene el fichero:

LABEL RECORDS ARE STANDARD

para todos, excepto los ficheros asignados a una impresora, que deben llevar:

LABEL RECORDS ARE OMITTED

El nombre se debe corresponder con el definido en la cláusula SELECT.

Antes de explicar la cláusula BLOCK CONTAINS es necesario aclarar el concepto de bloque: un bloque o registro físico es la unidad de información que un ordenador puede leer o escribir. Su longitud depende del modelo de ordenador.

Por el contrario, un registro lógico es una unidad de información definida por el programador. Un registro físico puede estar constituido por varios registros lógicos y en ese caso se leerán y/o se escribirán todos a la vez. Esta situación es transparente para el programador, ya que es el sistema operativo el encargado de gestionarla, pero si los registros se van a agrupar en bloques el compilador debe saberlo.

BLOCK CONTAINS N RECORDS

Donde N es el número de registros lógicos; también es conocido como factor de bloqueo.

Si N = 1 esta cláusula puede ser omitida, y significa que el registro físico y el lógico coinciden.

Aunque lo usual es que todos los registros de un fichero tengan la misma longitud, el COBOL tiene prevista una instrucción en el caso de que esto no suceda.

RECORDING MODE IS { F | U | V }

F: se trata de registros de longitud fija. Es la opción tomada por defecto.

U: los registros tienen distinta longitud, pero pueden agruparse en bloques.

V: los registros son de distinta longitud, pero no pueden bloquearse.

Cuando los registros son de longitud variable, se pueden emplear dos instrucciones adicionales con el fin de proporcionar más información al compilador:

RECORD CONTAINS N CHARACTERS

que especifica el tamaño máximo de los registros. Si se omite, el compilador explora la descripción de los registros para conocerlo.

$$\underline{\text{DATA}} \left\{ \begin{array}{l} \underline{\text{RECORD}} \text{ IS} \\ \underline{\text{RECORDS}} \text{ ARE} \end{array} \right\} \text{nombre-registro-1 ...}$$

Y permite dar un nombre distinto a cada uno de los registros lógicos diferentes contenidos en el fichero. Apenas se utiliza.

A continuación de la cláusula **FD** debe aparecer la descripción de la estructura del registro. Para ello se utiliza un campo descompuesto en subcampos:

```
01 NOMBRE-REGISTRO.
   05 CAMPO-1 ... PIC
   .
   .
   .
   05 CAMPO-K ... PIC
```

También se puede definir solamente la longitud y utilizar la **WORKING-STORAGE SECTION** para realizar la subdivisión:

```
                    01 NOMBRE-REGISTRO PIC X(N).
                                             .
                                             .
                                             .
WORKING-STORAGE SECTION.
    01 NOMBRE-REGISTRO-W.
       05 CAMPO-1 ... PIC
       .
       .
       .
       05 CAMPO-K ... PIC
```

Se verá más adelante cómo estas dos posibilidades ocasionan instrucciones de lectura/escritura diferentes.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-FICHEROS.


ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PC.
   OBJECT-COMPUTER. IBM-PC.
   SPECIAL-NAMES.
       DECIMAL-POINT IS COMMA.
```

```
  INPUT-OUTPUT SECTION.

    FILE-CONTROL.

        SELECT CLIENTE ASSIGN TO DISK
                       ORGANIZATION SEQUENTIAL
                       ACCESS SEQUENTIAL.

        SELECT IMPRESO ASSIGN TO PRINTER
                       ORGANIZATION SEQUENTIAL
                       ACCESS SEQUENTIAL.

 DATA DIVISION.

  FILE SECTION.

   FD CLIENTE LABEL RECORD STANDARD
              VALUE OF FILE-ID 'CLIENTE.DAT'.

*
* La clausula VALUE OF FILE-ID 'NOMBRE FICHERO' es propia del
* COBOL de MICROSOFT y relaciona el nombre físico del fichero
* con el nombre lógico.
* Se entiende por nombre físico el nombre que tiene en el disco
* el fichero.
*

   01 REG-CLIENTE.
      05 CODIGO-CLIENTE           PIC 9(4).
      05 NOMBRE-CLIENTE           PIC X(30).
      05 DIRECC-CLIENTE           PIC X(50).
      05 TOTAL-FACTURA            PIC 9(9).
      05 DESCUENTO                PIC 99V99.

   FD IMPRESO LABEL RECORD OMITTED.

   01 LINEA                       PIC X(80).
```

Se trata de la primera parte de un programa: inicialmente sus instrucciones de definición: dos ficheros, uno de ellos asociado a un disco y el otro a una impresora.

OPERACIONES CON FICHEROS 14

UNA vez definidos todos los ficheros de entrada y/o salida con los que va a trabajar el programa, así como los dispositivos en que se encuentran, es necesario conocer las instrucciones necesarias para operar con ellos.

OPEN

Es la primera operación y resulta imprescindible, ya que su misión es situar las cabezas de lectura o escritura al comienzo del fichero.

El formato de la instrucción y su significado se explican a continuación: can a continuación:

```
OPEN[INPUT nombre-de-fichero]
    [OUTPUT nombre-de-fichero]
    [I-O nombre-de-fichero]
```

Se emplea:

— OPEN INPUT. Si el fichero se va a utilizar únicamente como entrada: en él no se va a escribir, sólo se leerán sus registros.

— OPEN OUTPUT. El fichero se crea como salida; en él sólo se podrá grabar. Si el fichero existiera con anterioridad y tuviera contenido, éste se destruye y el fichero aparecerá vacío. Un típico ejemplo de un fichero abierto sólo como salida es uno de impresión.

— OPEN I-O. A lo largo del programa se va a leer y a escribir en el fichero. Un ejemplo es un fichero que sufre un proceso de actualización.

Es necesario tener presente que al abrir un fichero mediante una operación OPEN no se procesa ningún registro: sólo se coloca al comienzo del mismo.



READ

Si un fichero ha sido abierto como INPUT o I-O, sus registros van a ser leídos a lo largo del programa. Esta lectura se realiza mediante una operación READ, que transfiere a las áreas de trabajo el contenido del registro.

Si el contenido del registro en el fichero es:

0010ISIDORO SORIA SANZ 9300012

y se ha definido una estructura de registro de la forma:

```
01 REGISTRO.
   05 COD-CLIENTE PIC 9(4).
   05 NOM-CLIENTE PIC X(20).
   05 TOT-CLIENTE PIC 9(5).
   05 IVA-CLIENTE PIC 99.
```

Después de una instrucción de lectura, cuyo formato se explicará a continuación, los campos englobados en REGISTRO tendrán los valores:

COD-CLIENTE = 0010
NOM-CLIENTE = ISIDORO SORIA SANZ
TOT-CLIENTE = 93000
IVA-CLIENTE = 12

A continuación ya se podrán efectuar operaciones con los campos recién «rellenados» por medio de la lectura. Se podría decir que una operación de lectura transfiere el contenido de un registro en un fichero a un campo de datos definido en el programa.

Unicamente se va a explicar una instrucción de lectura, que se corresponde con un acceso secuencial; el formato es:

READ nombre-de-fichero [INTO campo]
AT END operación.

Ya se ha hecho suficiente hincapié en el significado de esta instrucción, que aporta dos nociones nuevas:

La opción INTO campo posibilita al programador definir la estructura del registro en la WORKING-STORAGE SECTION en lugar de hacer esta definición en la cláusula FD de la DATA DIVISION.

Las definiciones de ficheros de las siguientes figuras son equivalentes:

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-LECTURA.


ENVIRONMENT DIVISION.
```

```
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PC.
   OBJECT-COMPUTER. IBM-PC.
   SPECIAL-NAMES.
       DECIMAL-POINT IS COMMA.

 INPUT-OUTPUT SECTION.

   FILE-CONTROL.

       SELECT FICHERO ASSIGN TO DISK
                      ORGANIZATION LINE SEQUENTIAL
                      ACCESS MODE SEQUENTIAL.


DATA DIVISION.

 FILE SECTION.

  FD FICHERO LABEL RECORD STANDARD
             VALUE OF FILE-ID 'CLIENTE.DAT'.

  01 REGISTRO                      PIC X(31).

 WORKING-STORAGE SECTION.

  01 REGISTRO-W.
     05 COD-CLIENTE                PIC 9(4).
     05 NOM-CLIENTE                PIC X(20).
     05 TOT-CLIENTE                PIC 9(5).
     05 IVA-CLIENTE                PIC 99.

  01 TOTAL-FACTURA                 PIC 9(6),99.
  01 FINAL-FICHERO                 PIC X.


PROCEDURE DIVISION.

PRINCIPAL.
       OPEN INPUT FICHERO.
       MOVE 'N' TO FINAL-FICHERO.
       PERFORM LEER-FICHERO.
       PERFORM TRATAMIENTO UNTIL FINAL-FICHERO = 'S'.
       CLOSE FICHERO.
       STOP RUN.

LEER-FICHERO.
       READ FICHERO INTO REGISTRO-W
       AT END MOVE 'S' TO FINAL-FICHERO.

TRATAMIENTO.
       COMPUTE TOTAL-FACTURA = TOT-CLIENTE +
                               TOT-CLIENTE * IVA-CLIENTE / 100.
       DISPLAY 'CODIGO CLIENTE: ' COD-CLIENTE.
       DISPLAY 'NOMBRE: ' NOM-CLIENTE.
       DISPLAY 'TOTAL: ' TOTAL-FACTURA.
       PERFORM LEER-FICHERO.
```

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-LECTURA.

ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PC.
   OBJECT-COMPUTER. IBM-PC.
   SPECIAL-NAMES.
         DECIMAL-POINT IS COMMA.

 INPUT-OUTPUT SECTION.

   FILE-CONTROL.

         SELECT FICHERO ASSIGN TO DISK
                             ORGANIZATION LINE SEQUENTIAL
                             ACCESS MODE SEQUENTIAL.

DATA DIVISION.

 FILE SECTION.

  FD FICHERO LABEL RECORD STANDARD
             VALUE OF FILE-ID 'CLIENTE.DAT'.

  01 REGISTRO.
     05 COD-CLIENTE                  PIC 9(4).
     05 NOM-CLIENTE                  PIC X(20).
     05 TOT-CLIENTE                  PIC 9(5).
     05 IVA-CLIENTE                  PIC 99.

 WORKING-STORAGE SECTION.

  01 TOTAL-FACTURA                   PIC 9(6),99.
  01 FINAL-FICHERO                   PIC X.

PROCEDURE DIVISION.

PRINCIPAL.
         OPEN INPUT FICHERO.
         MOVE 'N' TO FINAL-FICHERO.
         PERFORM LEER-FICHERO.
         PERFORM TRATAMIENTO UNTIL FINAL-FICHERO = 'S'.
         CLOSE FICHERO.
         STOP RUN.

LEER-FICHERO.
         READ FICHERO AT END MOVE 'S' TO FINAL-FICHERO.

TRATAMIENTO.
         COMPUTE TOTAL-FACTURA = TOT-CLIENTE +
                                 TOT-CLIENTE * IVA-CLIENTE / 100.
         DISPLAY 'CODIGO CLIENTE: ' COD-CLIENTE.
         DISPLAY 'NOMBRE: ' NOM-CLIENTE.
         DISPLAY 'TOTAL: ' TOTAL-FACTURA.
         PERFORM LEER-FICHERO.
```

La única diferencia existente entre ellos es la manera de definir y leer el registro.

En la segunda la definición de la estructura del registro se hace en la FILE SECTION y, por tanto, se lee READ FICHERO; en el primer programa en la FILE SECTION se define únicamente la longitud del registro (01 REGISTRO PIC X(31)), mientras que la estructura del registro se describe en la WORKING.

En la instrucción de lectura, READ FICHERO INTO REGISTRO-W, se especifica el campo de la WORKING. Todas las operaciones posteriores se harán con los campos en que se subdivide REGISTRO-W.

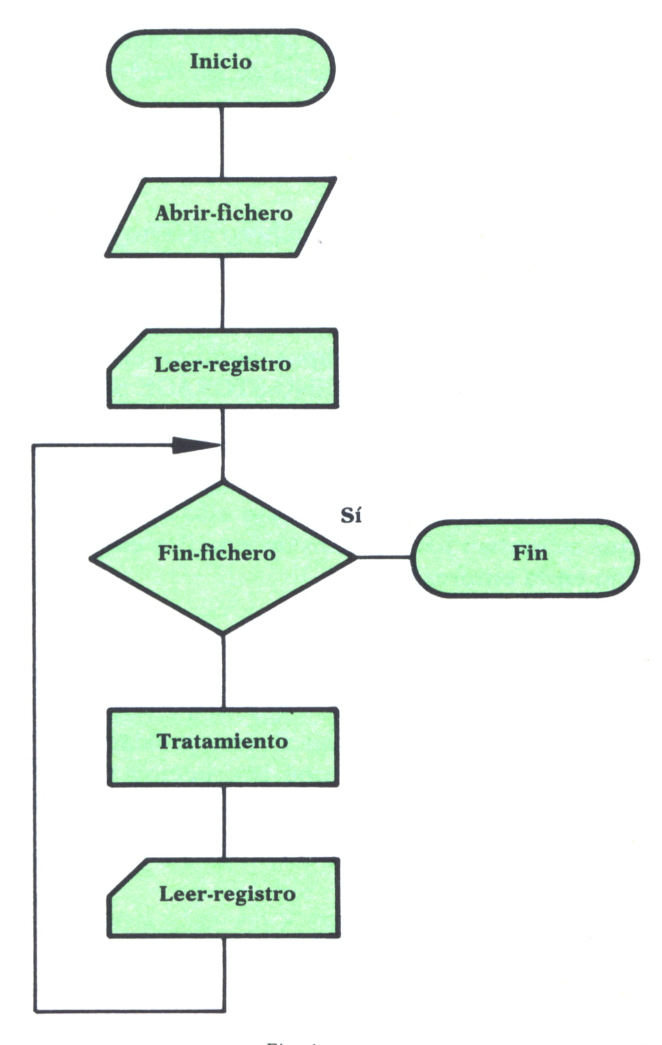


Fig. 1.

Se puede ya observar que la instrucción READ FICHERO INTO es equivalente a:

READ FICHERO
MOVE REGISTRO TO REGISTRO-W.

En el programa aparece también la instrucción AT END, que equivale a una instrucción IF: si se detecta un final de fichero al leer un registro se realizan las operaciones que se indican a continuación. En este caso se pone a «S» el campo FINAL-FICHERO.

La estructura de los dos programas es igual y es muy conocida en el mundo de las aplicaciones: se trata de la técnica de lectura anticipada; antes de comenzar el bucle principal se lee un registro. De esta manera, si el fichero esta vacío, nunca se ejecuta dicho bucle.



WRITE

Permite grabar registros en un fichero abierto con un OPEN OUTPUT y con organización secuencial: su formato es muy sencillo:

WRITE nombre-de-registro [FROM campo].

Traslada los valores de los campos definidos en el programa (ya sea en la FILE SECTION o en la WORKING-STORAGE SECTION si se utiliza la opción FROM campo) al registro del fichero.

Su utilización es tan sencilla que no se considera necesario ampliar su explicación. Unicamente se añade un ejemplo: se trata de la lectura de un fichero maestro y su actualización. Para ello se emplea otro fichero nuevo, en el que se graban los registros de aquellos empleados que tengan un contrato fijo en la empresa.



CLOSE

Todos los ficheros que se hayan abierto previamente con una instrucción OPEN deben cerrarse con una instrucción CLOSE. Su formato se muestra seguidamente y su utilización se ilustra en los programas anteriores.

CLOSE nombre-de-fichero.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-LECTURA.

ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-PC.
   OBJECT-COMPUTER. IBM-PC.
   SPECIAL-NAMES.
       DECIMAL-POINT IS COMMA.

 INPUT-OUTPUT SECTION.

   FILE-CONTROL.

       SELECT FICHERO ASSIGN TO DISK
                      ORGANIZATION LINE SEQUENTIAL
                      ACCESS MODE SEQUENTIAL.

DATA DIVISION.

 FILE SECTION.

  FD FICHERO LABEL RECORD STANDARD
             VALUE OF FILE-ID 'CLIENTE.DAT'.

  01 REGISTRO                         PIC X(31).

 WORKING-STORAGE SECTION.

  01 REGISTRO-W.
     05 COD-CLIENTE                   PIC 9(4).
     05 NOM-CLIENTE                   PIC X(20).
     05 TOT-CLIENTE                   PIC 9(5).
     05 IVA-CLIENTE                   PIC 99.

  01 TOTAL-FACTURA                    PIC 9(6),99.
  01 FINAL-FICHERO                    PIC X.

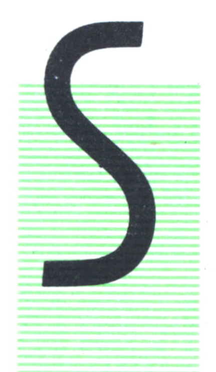
PROCEDURE DIVISION.

PRINCIPAL.
       OPEN INPUT FICHERO.
       MOVE 'N' TO FINAL-FICHERO.
       PERFORM LEER-FICHERO.
       PERFORM TRATAMIENTO UNTIL FINAL-FICHERO = 'S'.
       CLOSE FICHERO.
       STOP RUN.

LEER-FICHERO.
       READ FICHERO INTO REGISTRO-W
       AT END MOVE 'S' TO FINAL-FICHERO.

TRATAMIENTO.
       COMPUTE TOTAL-FACTURA = TOT-CLIENTE +
                               TOT-CLIENTE * IVA-CLIENTE / 100.
       DISPLAY 'CODIGO CLIENTE: ' COD-CLIENTE.
       DISPLAY 'NOMBRE: ' NOM-CLIENTE.
       DISPLAY 'TOTAL: ' TOTAL-FACTURA.
       PERFORM LEER-FICHERO.
```

LISTADOS 15



ON un caso particular de ficheros, por lo que se considera conveniente dedicarles un apartado especial.

Se trata de ficheros intrínsecamente secuenciales y asociados a una impresora, por lo que la cláusula **SELECT** debe ser:

SELECT LISTADO ASSIGN TO PRINTER
ORGANIZACION SEQUENTIAL.

En muchos compiladores es posible omitir, incluso, la cláusula **ORGANIZATION**, cuando se trata de ficheros de impresora.

La longitud más usual de un registro de estas características es de 80 ó 132 caracteres;

FD LISTADO LABEL RECORDS OMITTED.
01 LINEA PIC X(132) o X(80).

Puesto que en un mismo listado hay diferentes tipos de líneas, todas ellas de la misma longitud, pero de diferentes contenidos, es preferible no definirlas todas en la **FILE SECTION** y hacerlo en la **WORKING-STORAGE**, empleando, por tanto, como operación de escritura un **WRITE LINEA FROM...**

Lógicamente se debe abrir como salida: **OPEN OUTPUT LISTADO.**

Pero este tipo de ficheros tienen una instrucción de escritura particular que le permite dejar líneas en blanco y hacer saltos de página de una forma cómoda y sencilla:

WRITE nombre-de-registro [FROM identificador]

[{BEFORE / AFTER} ADVANCING {N LINES / PAGE}]

El significado de **BEFORE** es que la línea se escribirá antes de saltar de página o dejar líneas en blanco, mientras que con **AFTER** primero se realiza el espaciado y seguidamente se imprime la línea.

Un ejemplo clarificador de este tipo de ficheros se encuentra en el programa siguiente:

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. MAESTRO.

*
*   Una empresa desea, al final de año actualizar el fichero
* general de empleados.
*   Para ello dispone de un fichero "MAESTRO", que contiene un
* registro por cada empleado, con datos como número de nómina,
* sueldo base, IRPF, etc. Además contiene un campo especial, que
* indica si el empleado tiene un contrato fijo o eventual: si la
* fecha grabada en él (AAMM) es mayor que la fecha actual, el
* empleado sigue perteneciendo a la empresa y su registro se
* grabará en un nuevo fichero (ACTUAL).
*   Se desea obtener un listado de todos aquellos empleados
* cuyo contrato ha vencido.
*   En el listado debe aparecer también el sueldo correspondiente
* al último mes, sin considerar la seguridad social y añadiendo
* una prima por número de hijos (1% por cada uno).
*

ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

  SOURCE-COMPUTER. IBM-PC.
  OBJECT-COMPUTER. IBM-PC.
  SPECIAL-NAMES.
       DECIMAL-POINT IS COMMA.

 INPUT-OUTPUT SECTION.

  FILE-CONTROL.

       SELECT MAESTRO ASSIGN TO DISK
                      ORGANIZATION IS LINE SEQUENTIAL.

       SELECT ACTUAL  ASSIGN TO DISK
                      ORGANIZATION IS LINE SEQUENTIAL.

       SELECT LISTADO ASSIGN TO PRINTER.

DATA DIVISION.

 FILE SECTION.

  FD MAESTRO LABEL RECORD STANDARD
             VALUE OF FILE-ID 'MAESTRO.DAT'.

  01 REG-MAESTRO                  PIC X(100).

  FD ACTUAL  LABEL RECORD STANDARD
             VALUE OF FILE-ID 'ACTUAL.DAT'.

  01 REG-ACTUAL                   PIC X(100).
```

```
 FD LISTADO LABEL RECORD OMITTED.

 01 LINEA                        PIC X(80).

WORKING-STORAGE SECTION.
*
*   Puesto que ambos ficheros tienen una misma estructura
* de registro, es mejos definir el registro en la WORKING
* y emplear las clausulas READ MAESTRO INTO REG-COMUN y
* WRITE REG-ACTUAL FROM REG-COMUN.
*

   01 REG-COMUN.
      05 NUM-NOMINA               PIC X(6).
      05 NOMBRE                   PIC X(30).
      05 DIRECCION                PIC X(50).
      05 SUELDO-BASE              PIC 9(6).
      05 NUM-HIJOS                PIC 99.
      05 IRPF                     PIC 99.
      05 FEC-VENCIMIENTO          PIC 9(4).

   01 FINAL-FICHERO               PIC X          VALUE 'N'.
   01 CONT-PAGINAS                PIC 99         VALUE ZERO.

*   Campo destinado a controlar el salto de página. Cuando se
* hayan impreso más de 50 líneas se saltará de página y se
* escribirán nuevamente las cabeceras. Por eso toma como valor
* inicial 51.

   01 CONT-LINEAS                 PIC 99         VALUE 51.

*   Campo que almacenará el número de empleados en plantilla.
* Los empleados con contrato fijo tienen una fecha de
* vencimiento toda a 9's (9999).
*

   01 CONTADOR                    PIC 9(6)       VALUE ZERO.

*
*   Campo intermedio que sirve para calcular el % del
* sueldo base y que permite simplificar las operaciones
* posteriores.
*

   01 PORCENTAJE                  PIC 9(4)V99.

*
*   Campo que almacenará la fecha de proceso. La fecha del
* sistema tiene formato AAMMDD y para efectuar la comparacion
* se recibe en un campo subdivido en dos partes: La primera
* contendrá el año y el mes y la segunda el dia.
*

   01 FECHA.
      05 COMPARAR                 PIC 9(4).
      05 DIA                      PIC 99.

*
*   Definición de las líneas del informe.
```

```
*
     01 CABECERA1.
        05 FILLER                      PIC X(57)         VALUE SPACES.
        05 FILLER                      PIC X(8)          VALUE
                                       'PAGINA:
        05 PAGINA-CAB1                 PIC Z9.

     01 CABECERA2.
        05 FILLER                      PIC X(20)         VALUE SPACES.
        05 FILLER                      PIC X(32)         VALUE
           'PERSONAL EN PLANTILLA CON FECHA '.
        05 FECHA-CAB2                  PIC 99/99.

     01 SUBRAYAR2.
        05 FILLER                      PIC X(20)         VALUE SPACES.
        05 FILLER                      PIC X(37)         VALUE ALL '-'.

     01 CABECERA3.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 FILLER                      PIC X(6)          VALUE 'NOMINA'.
        05 FILLER                      PIC X(10)         VALUE SPACES.
        05 FILLER                      PIC X(18)         VALUE
           'NOMBRE Y APELLIDOS'.
        05 FILLER                      PIC X(10)         VALUE SPACES.
        05 FILLER                      PIC X(6)          VALUE 'SUELDO'.
        05 FILLER                      PIC X(5)          VALUE SPACES.
        05 FILLER                      PIC XX            VALUE 'NH'.
        05 FILLER                      PIC X(3)          VALUE SPACES.
        05 FILLER                      PIC X(4)          VALUE 'IRPF'.
        05 FILLER                      PIC X(7)          VALUE SPACES.
        05 FILLER                      PIC X(5)          VALUE 'TOTAL'.

     01 SUBRAYAR3.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 FILLER                      PIC X(6)          VALUE ALL '-'.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 FILLER                      PIC X(30)         VALUE ALL '-'.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 FILLER                      PIC X(6)          VALUE ALL '-'.
        05 FILLER                      PIC X(5)          VALUE SPACES.
        05 FILLER                      PIC XX            VALUE '--'.
        05 FILLER                      PIC X(3)          VALUE SPACES.
        05 FILLER                      PIC X(4)          VALUE ALL '-'.
        05 FILLER                      PIC X(7)          VALUE SPACES.
        05 FILLER                      PIC X(5)          VALUE ALL '-'.

     01 LINEA-DETALLE.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 NOMINA-LINEA                PIC X(6).
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 NOMBRE-LINEA                PIC X(30).
        05 FILLER                      PIC X(3)          VALUE SPACES.
        05 SUELDO-LINEA                PIC Z(3).9(3).
        05 FILLER                      PIC X(5)          VALUE SPACES.
        05 HIJOS-LINEA                 PIC Z9.
        05 FILLER                      PIC X(4)          VALUE SPACES.
        05 IRPF-LINEA                  PIC 99.
        05 FILLER                      PIC X(4)          VALUE SPACES.
```

```
     05 TOTAL                    PIC Z.Z(3).9(3).

 PROCEDURE DIVISION.

*
*   En él se efectúan las llamadas a los restantes procedimientos
* del programa. Al estar todo centralizado y estructurado, resulta
* un programa de más fácil lectura.
*

 PRINCIPAL.
        PERFORM ABRIR-FICHEROS.
        PERFORM LEER-MAESTRO.
        PERFORM TRATAMIENTO UNTIL FINAL-FICHERO = 'S'.
        PERFORM CERRAR-FICHEROS.
        STOP RUN.
 TRATAMIENTO.

*   Primeramente se comprueba si se trata de un empleado con
* contrato vigente. Si es así se grabará dará de alta el
* registro en el fichero ACTUAL.
*   Si ha finalizado el contrato se imprimirá la correspondiente
* la correspondiente línea de fichero.

        IF FEC-VENCIMIENTO > COMPARAR
           PERFORM GRABAR-ACTUAL
           ADD 1  TO CONTADOR
        ELSE
           PERFORM ESCRIBIR-INFORMES.
        PERFORM LEER-MAESTRO.

 ESCRIBIR-INFORMES.

*   Antes de escribir la linea de detalle se comprueba si
* ya se han escrito 50 líneas. Si es así se imprimirán las
* cabeceras. Por este motivo y para que las cabeceras se
* escriban la primera vez, el contador de líneas está
* inicializado a 51.
*   Después se calculan los valores a imprimir y se mueven
* a la línea de detalle, para a continuación imprimir dicha
* línea WRITE LINEA FROM LINEA-DETALLE.

        IF CONT-LINEAS > 50
           PERFORM ESCRIBIR-CABECERAS.
        COMPUTE PORCENTAJE = SUELDO-BASE / 100.
        COMPUTE TOTAL = SUELDO-BASE + PORCENTAJE * NUM-HIJOS
                                    - PORCENTAJE * IRPF.
        MOVE NUM-NOMINA    TO NOMINA-LINEA.
        MOVE NOMBRE        TO NOMBRE-LINEA.
        MOVE NUM-HIJOS     TO HIJOS-LINEA.
        MOVE IRPF          TO IRPF-LINEA.
        MOVE SUELDO-BASE   TO SUELDO-LINEA.
        WRITE LINEA FROM LINEA-DETALLE AFTER 1.
        ADD 1 TO CONT-LINEAS.

 ABRIR-FICHEROS.

*   MAESTRO sólo se utiliza para lectura, se abre com INPUT.
```

```
*    En ACTUAL se van a grabar registros: se abre como OUTPUT.
*    Lógicamente la impresora se abre como salida.

          OPEN INPUT  MAESTRO.
          OPEN OUTPUT ACTUAL
                      LISTADO.

*    Mediante un ACCEPT se lee la fecha del día desde el sistema.

          ACCEPT FECHA FROM DATE.

 LEER-MAESTRO.

*    El programa finalizará cuando se hayan leído todos los
* registros del fichero MAESTRO. Se controla con la clausula
* AT END.

          READ MAESTRO INTO REG-COMUN
          AT END MOVE 'S' TO FINAL-FICHERO.

 GRABAR-ACTUAL.

*    Los registros del fichero ACTUAL se graban desde el campo
* de WORKING REG-COMUN.
          WRITE REG-ACTUAL FROM REG-COMUN.

 CERRAR-FICHEROS.

          CLOSE  MAESTRO
                 ACTUAL
                 LISTADO.
          DISPLAY 'EL NUMERO DE EMPLEADOS EN PLANTILLA ES: '
                   CONTADOR.


 ESCRIBIR-CABECERAS.

          ADD 1 TO CONT-PAGINAS.
          MOVE CONT-PAGINAS TO PAGINA-CAB1.
          MOVE COMPARAR     TO FECHA-CAB2.
          WRITE LINEA FROM CABECERA1 AFTER PAGE.
          WRITE LINEA FROM CABECERA2 AFTER 2.
          WRITE LINEA FROM SUBRAYAR2 AFTER 1.
          WRITE LINEA FROM CABECERA3 AFTER 2.
          WRITE LINEA FROM SUBRAYAR3 AFTER 1.
          MOVE 7            TO CONT-LINEAS.
```

Se pueden observar en este programa, suficientemente documentado y explicativo, dos aspectos importantes; dos campos CONT-LINEAS y CONT-PAGINAS muy usuales en estos programas de impresión.

APENDICE

PROGRAMA DE ALTAS, BAJAS Y MODIFICACIONES

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID. ACTUALIZAR.

*   PROGRAMA QUE A PARTIR DE UN FICHERO MAESTRO Y OTRO DE
*   MOVIMIENTO OBTIENE EL NUEVO MAESTRO. EN EL FICHERO DE
*   MOVIMIENTO SE REFLEJAN ALTAS, MODIFICACIONES Y BAJAS CON UN
*   CODIGO.
*   LOS TRES FICHEROS ESTAN ORDENADOS POR CODIGO DE EMPLEADO.
*   AL MISMO TIEMPO QUE SE ACTUALIZA SE OBTIENE UN LISTADO CON
*   LOS POSIBLES ERRORES QUE SE PRODUZCAN.
*   POR ULTIMO, SE ELABORA UN LISTADO CON EL CONTENIDO FINAL DEL
*   FICHERO MAESTRO.

 ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
   FILE-CONTROL.

           SELECT VIEJO ASSIGN TO DISK
                        ORGANIZATION IS LINE SEQUENTIAL.

           SELECT NUEVO ASSIGN TO DISK
                        ORGANIZATION IS LINE SEQUENTIAL.

           SELECT MOVI  ASSIGN TO DISK
                        ORGANIZATION IS LINE SEQUENTIAL.

           SELECT IMPRE ASSIGN TO PRINTER.

 DATA DIVISION.
  FILE SECTION.

   FD  VIEJO LABEL RECORDS STANDARD
             VALUE OF FILE-ID 'VIEJO.DAT'.

   01  REG-VIEJO.
       05 COD-EMP-V              PIC X(5).
       05 NOMBRE-V               PIC X(33).
       05 DIREC-V                PIC X(25).
       05 TLF-V                  PIC X(11).

   FD  NUEVO LABEL RECORDS STANDARD
             VALUE OF FILE-ID 'NUEVO.DAT'.

   01  REG-NUEVO.
       05 COD-EMP-N              PIC X(5).
       05 NOMBRE-N               PIC X(33).
       05 DIREC-N                PIC X(25).
       05 TLF-N                  PIC X(11).

   FD  MOVI  LABEL RECORDS STANDARD
             VALUE OF FILE-ID 'MOVI.DAT'.

   01  REG-MOVI.
       05 DATOS.
          10 COD-EMP-M           PIC X(5).
          10 NOMBRE-M            PIC X(33).
          10 DIREC-M             PIC X(25).
          10 TLF-M               PIC X(11).
       05 CODIGO-M               PIC X.

   FD  IMPRE LABEL RECORDS OMITTED.

   01  LINEA                      PIC X(80).
```

```
WORKING-STORAGE SECTION.

01 CON-LIN                          PIC 99      VALUE 56.
01 FIN-NUEVO                        PIC X       VALUE 'N'.

01 CABECERA-1.
   05 FILLER                        PIC X(20) VALUE SPACES.
   05 FILLER                        PIC X(22) VALUE
                                        'LISTADO DE ERRORES'.

01 SUBRAYADO-1.
   05 FILLER                        PIC X(20) VALUE SPACES.
   05 FILLER                        PIC X(22) VALUE ALL '-'.

01 LINEA-ERRORES.
   05 FILLER                        PIC X(2)  VALUE SPACES.
   05 COD-EMP-E                     PIC X(5).
   05 FILLER                        PIC X(2)  VALUE SPACES.
   05 TEXTO-E                       PIC X(50).

01 CABECERA-2.
   05 FILLER                        PIC X(20) VALUE SPACES.
   05 FILLER                        PIC X(21) VALUE
                                        'LISTADO NUEVO MAESTRO'.

01 SUBRAYADO-2.
   05 FILLER                        PIC X(20) VALUE SPACES.
   05 FILLER                        PIC X(21) VALUE ALL '-'.

01 DETALLE.
   05 FILLER                        PIC X(2)  VALUE SPACES.
   05 CODIGO                        PIC X(5).
   05 FILLER                        PIC X(2)  VALUE SPACES.
   05 NOMBRE                        PIC X(33).
   05 FILLER                        PIC X(2)  VALUE SPACES.
   05 DIRECCION                     PIC X(25).
   05 FILLER                        PIC X(2).
   05 TELEFONO                      PIC X(11).

01 BLANCOS                          PIC X(80) VALUE SPACES.


PROCEDURE DIVISION.

 INICIO.
           PERFORM CREAR-NUEVO.
           PERFORM LECTURA-NUEVO.

 FINAL-PROGRAMA.
           STOP RUN.

 CREAR-NUEVO.
           OPEN INPUT  VIEJO MOVI
                OUTPUT NUEVO IMPRE.
           PERFORM LEER-VIEJO.
           PERFORM LEER-MOVI.
           PERFORM ACTUALIZAR UNTIL COD-EMP-V = HIGH-VALUES AND
                                    COD-EMP-M = HIGH-VALUES.
           CLOSE VIEJO MOVI NUEVO.

 LECTURA-NUEVO.
           OPEN INPUT  NUEVO.
           MOVE 56 TO CON-LIN.
           PERFORM LEER-NUEVO.
           PERFORM ESCRIBIR UNTIL FIN-NUEVO = 'S'.
           CLOSE NUEVO IMPRE.

 ACTUALIZAR.
           IF COD-EMP-V < COD-EMP-M
              PERFORM DUPLICAR
           ELSE
              IF COD-EMP-V = COD-EMP-M
                 PERFORM TRATAR-CODIGO
              ELSE
                 IF CODIGO-M = 'A'
                    PERFORM ALTA
                 ELSE
                    PERFORM TRATAR-ERROR.
```

```
DUPLICAR.
        MOVE REG-VIEJO TO REG-NUEVO.
        WRITE REG-NUEVO.
        PERFORM LEER-VIEJO.

TRATAR-CODIGO.
        IF CODIGO-M = 'M'
           MOVE REG-MOVI TO REG-NUEVO
           WRITE REG-NUEVO
        ELSE
           IF CODIGO-M ='A'
              PERFORM ERROR-ALTA.
        PERFORM LEER-VIEJO.
        PERFORM LEER-MOVI.

ALTA.
        MOVE REG-MOVI TO REG-NUEVO.
        WRITE REG-NUEVO.
        PERFORM LEER-MOVI.

ERROR-ALTA.
        MOVE COD-EMP-M TO COD-EMP-E.
        MOVE 'REGISTRO YA EXISTE' TO TEXTO-E.
        PERFORM CAMBIO-PAG1.
        WRITE LINEA FROM LINEA-ERRORES.

TRATAR-ERROR.
        MOVE COD-EMP-M TO COD-EMP-E.
        IF CODIGO-M = 'M'
           MOVE 'NO EXISTE REGISTRO A SER MODIFICADO' TO TEXTO-E
        ELSE
           MOVE 'NO EXISTE REGISTRO A SER DADO DE BAJA'
                 TO TEXTO-E.
        PERFORM CAMBIO-PAG1.
        WRITE LINEA FROM LINEA-ERRORES.
        PERFORM LEER-MOVI.

CAMBIO-PAG1.
        IF CON-LIN > 55
           WRITE LINEA FROM CABECERA-1 AFTER PAGE
           WRITE LINEA FROM SUBRAYADO-1
           WRITE LINEA FROM BLANCOS
           MOVE ZERO TO CON-LIN.
        ADD 1 TO CON-LIN.


ESCRIBIR.
         MOVE COD-EMP-N TO CODIGO.
         MOVE NOMBRE-N  TO NOMBRE.
         MOVE DIREC-N   TO DIRECCION.
         MOVE TLF-N     TO TELEFONO.
         PERFORM CAMBIO-PAG2.
         WRITE LINEA FROM DETALLE.
         PERFORM LEER-NUEVO.

CAMBIO-PAG2.
         IF CON-LIN > 55
            WRITE LINEA FROM CABECERA-2 AFTER PAGE
            WRITE LINEA FROM SUBRAYADO-2
            WRITE LINEA FROM BLANCOS
            MOVE ZERO TO CON-LIN.
         ADD 1 TO CON-LIN.

*  HIGH-VALUE ES UNA CONSTANTE FIGURATIVA QUE CONTIENE EL MAYOR
*  VALOR POSIBLE.
*  SE MUEVE ESTA CUANDO SE ENCUENTRA EL FINAL DE UN FICHERO
*  PARA QUE CONTINUE LEYENDO DEL OTRO FICHERO.

LEER-VIEJO.
         READ VIEJO AT END MOVE HIGH-VALUE TO COD-EMP-V.

LEER-MOVI.
         READ MOVI  AT END MOVE HIGH-VALUE TO COD-EMP-M.

LEER-NUEVO.
         READ NUEVO AT END MOVE 'S' TO FIN-NUEVO.
```

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

INDICE GENERAL

5 AÑADA PERIFERICOS A SU ORDENADOR

Breve descripción de varios periféricos que facilitan la comunicación con el ordenador personal, con algunos ejemplos de fácil construcción: ratón, lápiz óptico, marco para pantalla táctil...

6 GRAFICOS ANIMADOS CON EL ORDENADOR

En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comporender. Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc.

7 JUEGOS INTELIGENTES EN MICROORDENADORES

Los ordenadores pueden enfrentarse de forma «inteligente» ante puzzles y otros tipos de juegos. Esto es posible gracias al nuevo enfoque que ha dado la IA a la tradicional teoría de juegos.

8 PERIFERICOS INTERACTIVOS PARA SU ORDENADOR

Descripción detallada de la forma de construir, paso a paso y en su propia casa, dispositivos electrónicos que aumentarán la potencia y facilidad de uso de su ordenador: tableta digitalizadora, convertidores de señales analógicas, comunicaciones entre ordenadores.

9 COMO HACER DIBUJOS TRIDIMENSIONALES EN EL ORDENADOR PERSONAL

Compruebe que también con su ordenador personal puede llegar a diseñar y calcular imágenes en tres dimensiones con técnicas semejantes a las utilizadas por los profesionales del dibujo con equipos mucho más sofisticados.

10 PRACTIQUE MATEMATICAS Y ESTADISTICA CON EL ORDENADOR

En este libro se repasan los principales conceptos de las Matemáticas y la Estadística, desde un punto de vista eminentemente práctico y para su aplicación al ordenador personal. Se basan los diferentes textos en la presentación de pequeños programas (que usted podrá introducir en su ordenador personal).

11 CRIPTOGRAFIA: LA OCULTACION DE MENSAJES Y EL ORDENADOR

En este libro se presentan las técnicas de mensajes a través de la criptografía desde los primeros tiempos hasta la actualidad, en que el uso de los computadores ha proporcionado la herramienta necesaria para llegar al desarrollo de esta técnica.

12 APL: LENGUAJE PARA PROGRAMADORES DIFERENTES

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas y al mismo tiempo permite programar sin necesidad de conocer todos los elementos del lenguaje. Por ello es ideal para quienes reúnan imaginación y escasa formación en Informática.

13 ECONOMIA DOMESTICA CON EL ORDENADOR PERSONAL

Breve introducción a la contabilidad de doble partida y su aplicación al hogar, con explicaciones de cómo utilizar el ordenador personal para facilitar los cálculos, mediante un programa especialmente diseñado para ello.

14 COMO SIMULAR CIRCUITOS ELECTRONICOS EN EL ORDENADOR

Introducción a los diferentes métodos que se pueden emplear para simular y analizar circuitos electrónicos, mediante la utilización de diferentes lenguajes.

15 COMO CONSTRUIR SU PROPIO ORDENADOR

Cuando se trabaja con un ordenador, lo único que puede apreciarse, a simple vista, es una especie de caja negra que, misteriosamente, acepta una serie de instrucciones. En realidad, un ordenador es una máquina capaz de recibir, transformar, almacenar y suministrar datos.

16 EL ORDENADOR COMO INSTRUMENTO MUSICAL Y DE COMPOSICION

Análisis de cómo se puede utilizar el ordenador para la composición o interpretación de música. Libro eminentemente práctico, con numerosos ejemplos (que usted podrá practicar en su ordenador casero) y lleno de sugerencias para disfrutar haciendo de su ordenador un verdadero instrumento musical.

17 SISTEMAS OPERATIVOS: EL SISTEMA NERVIOSO DEL ORDENADOR

Características de diversos sistemas operativos utilizados en los ordenadores personales y caseros. Se trata de llegar al conocimiento, ameno aunque riguroso, de la misión del sistema operativo de su ordenador, para que usted consiga sacar mayor rendimiento a su equipo.

18 UNIX, EL ESTANDAR DE LOS SISTEMAS OPERATIVOS MULTIUSUARIO

La aparición y posterior difusión del sistema operativo UNIX supuso una revolución en el mercado, de tal modo que se ha convertido en el estándar de los sistemas multiusuario. Su aparente complejidad podría provocar, en principio, un primer rechazo, pero debido a su potencia se convierte rápidamente en una extraordinaria herramienta de trabajo apta para cualquier tipo de aplicaciones.

19 EL ORDENADOR Y LA ASTRONOMIA

Los cálculos astronómicos y el conocimiento del firmamento en un libro apasionante y curioso.

20 VISION ARTIFICIAL. TRATAMIENTO DE IMAGENES POR ORDENADOR

El procesado de imágenes es un campo de reciente y rápido desarrollo con importantes aplicaciones en área tan diversas como la mejora de imágenes biomédicas, robóticas, teledetección y otras aplicaciones industriales y militares. Se presentan los principios básicos, los sistemas y las técnicas de procesado más usuales.

21 PRACTIQUE HISTORIA Y GEOGRAFIA CON SU ORDENADOR

Libro interesante para los aficionados a estas ciencias, a quienes presenta una nueva visión de cómo utilizar el microordenador en su estudio.

22 LA CREATIVIDAD EN EL ORDENADOR. EXPERIENCIAS EN LOGO

El LOGO es un lenguaje enormemente capacitado para la creación principalmente gráfica y en especial para los niños. En este sentido se han desarrollado numerosas experiencias. En el libro se analizan estas experiencias y las posibilidades del LOGO en este sentido, así como su aplicación a su ordenador casero para que usted mismo (o con sus hijos) pueda repetirlas.

23 EL LENGUAJE C, PROXIMO A LA MAQUINA

Lenguaje de programación que se está imponiendo en los microordenadores más grandes, tanto por su facilidad de aprendizaje y uso, como por su enorme potencia y su adecuación a la programación estructurada. Vinculado íntimamente al sistema operativo UNIX es uno de los lenguajes de más futuro entre los que se utilizan los micros personales.

24 BASIC

El lenguaje BASIC es la forma más fácil de aprender las instrucciones más elementales con las que podemos mandar a nuestro ordenador que haga las más diversas tareas.

25 COMO ELEGIR UNA HOJA ELECTRONICA DE CALCULO

En este título se estudian las diferentes versiones existentes de esta aplicación típica, desde el punto de vista de su utilidad para, en función de las necesidades de cada usuario y del ordenador de que dispone, poder elegir aquella que más se adecúe a cada paso.

26 BASIC AVANZADO

Una vez conocidas las instrucciones fundamentales del lenguaje BASIC se plantea la cuestión de la realización de programas que resuelvan problemas o aplicaciones que se nos presentan diariamente en el trabajo, en casa o en los estudios. Este libro trata de mostrar cómo se podrían realizar algunas de estas aplicaciones, estudiando diversas estructuras que proporciona el lenguaje BASIC (como las subrutinas) y viendo las ideas fundamentales para realizar gráficos en pantalla mediante un programa y para almacenar datos en discos o cintas mediante los ficheros.

27 APLIQUE SU ORDENADOR A LAS CIENCIAS NATURALES

Ejemplos sencillos para practicar con el ordenador. Casos curiosos de la Naturaleza en forma de programas para su ordenador personal.

28 PRACTIQUE FISICA CON SU ORDENADOR

Deja que el ordenador te ayude en tus estudios. Materias tan difíciles como la Física, se ponen a tu alcance de una manera entretenida y mucho más clara, con programas que te permitirán entender las cosas desde un punto de vista más real.
Definiciones, fórmulas, gráficos y ejemplos, en un pequeño manual que puedas utilizar en cualquier momento.

29 PRACTIQUE QUIMICA CON SU ORDENADOR

En nuestra búsqueda particular de la «piedra filosofal», al modo de los antiguos alquimistas, nos ayudaremos del ordenador para que nos resulte más fácil. Con este libro conseguiremos entender fácilmente las valencias de los elementos, las reacciones Redox y las distintas teorías sobre el átomo. Nos servirá de guía para aprender la tabla periódica de los elementos y nos ayudará a comprender, mediante gráficos, una reacción en cadena. Podremos así convertir nuestra casa y nuestro ordenador en un gran laboratorio.

30 APRENDA MATEMATICAS Y ESTADISTICA CON EL LENGUAJE APL

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas. Indudablemente, es mucho más apto que BASIC para la construcción de pequeños programas que realicen operaciones matemáticas de dificultad media, que además se expresan de una forma muy semejante a la notación matemática ordinaria, lo que lo hace fácilmente comprensible.

31 LOS LENGUAJES DE LA INTELIGENCIA ARTIFICIAL

Libro en que se describen los lenguajes específicos para la «elaboración del saber» y los entornos de programación correspondientes. El conocimiento de estos lenguajes, además de interesante en sí mismo, es sumamente útil para entender todo lo que la Inteligencia Artificial supondrá para el futuro de la Informática.

32 LA ESTACION TERMINAL PERSONAL

Las modernas técnicas de comunicación van permitiendo que las grandes capacidades de proceso y el acceso a bases de datos de gran tamaño estén cada día más al alcance de cada usuario (fuera ya de los centros de proceso de datos).

33 COBOL

Este libro pretende introducir al lector en uno de los lenguajes más utilizados y menos considerados del mundo informático. El Cobol es el lenguaje de gestión por excelencia y está presente en el desarrollo del software en la gran mayoría de empresas e instituciones públicas.

34 ADA

El considerable esfuerzo desarrollado por el Departamento de Defensa de los Estados Unidos (DoD) para que el lenguaje Ada fuese desarrollado quedará compensado por las aportaciones de este lenguaje a los sistemas informáticos del futuro.
Sus aplicaciones originales, sistemas en tiempo real para mando y control en el área de Defensa, han sido ampliadas al campo industrial para el control de procesos, aplicaciones en tiempo real, inteligencia artificial, etcétera.

35 EL ORDENADOR Y LA LITERATURA

En este libro se examinan procesadores de textos, programas de análisis literario y una curiosa aplicación desarrollada por el autor: APOLO, un programa que compone estructuras poéticas.

36 EL ORDENADOR COMO MAQUINA DE ESCRIBIR INTELIGENTE

Descripción de algunos de los programas para tratamiento de textos existentes en el mercado, análisis comparativos y estudio de las posibilidades de cada uno de ellos. Guía práctica para la elección del procesador de textos que más se adecúe a nuestras necesidades y al ordenador personal del que dispongamos.

37 MS-DOS

El sistema operativo de muchos ordenadores personales es el sistema operativo de disco de Microsoft, más conocido como MS-DOS, que recibe su nombre de su principal actividad: manejar los discos y archivos de discos. Su conocimiento puede llegar a ser tan profundo como deseemos, las nociones básicas, sin embargo, pueden llegar a ser imprescindibles para el manejo de nuestro ordenador.

38 REDES DE AREA LOCAL

El objetivo de este libro es el de proporcionar al lector un conocimiento claro de lo que son las redes locales, de su tecnología, problemática y futuro, de forma que, si lo desea, pueda profundizar posteriormente, por medio de bibliografía especializada o por la práctica profesional.

39 LOS FUNDAMENTOS DE LA GRAFOLOGIA APLICADA Y SU POSIBLE TRATAMIENTO CON UN ORDENADOR PERSONAL

Se presentan en este libro los perfiles grafológicos óptimos correspondientes a diversas actividades laborales, así como los programas de ordenador necesarios para el manejo de estos datos. Obra eminentemente práctica y de aplicación de los conceptos teóricos desarrollados en ella.

40 ¿MAQUINAS MAS EXPERTAS QUE LOS HOMBRES?

Después de situar los «sistemas expertos» en el contexto de la Inteligencia Artificial y describir su construcción, su funcionamiento, su utilidad, etc., se analiza el papel que pueden tener en el futuro (y en el presente, ya) de la Informática, así como los polémicos temas de la «capacidad para desbancar a la inteligencia humana», y las posibilidades de «aprender» de que se puede dotar a un procesador, etcétera.

NOTA:
Ediciones Siglo Cultural, S. A., se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de esta colección.

Las pretensiones del presente libro son las de conseguir un acercamiento del lector hacia el muchas veces vilipendiado COBOL y dar a conocer el tipo de problemas que puede solventar de forma más rápida y sencilla que el resto de los lenguajes.

Este libro no es un manual de COBOL, es una iniciación a este lenguaje que muestra sucesivos, y cada vez más complejos, ejemplos de la estructura del mismo.

Los programas que se exponen en el libro han sido realizados con el compilador de COBOL de MICROSOFT versión 2.00 en un IBM/PC.